

# トランジスタ技術 SPECIAL

特集 PC9801と拡張インターフェースのすべて  
16ビット・パソコンを使いこなすためのハード&ソフト

No.3





A large crowd of people in formal attire dancing at a ball. The image is used as a background for the text.

# PARTNER

# WITH

例えば華やかに美しく  
ワルツを踊るためには  
知性溢れた  
とびっきりのパートナーが  
必要のように——。

# INTEREST

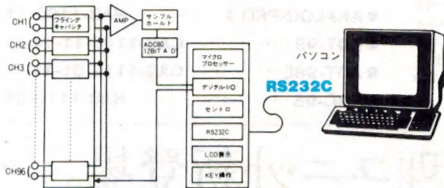
# GENGE.



# 機能性・拡張性を考えると パソコンのベストパートナーは PIUシリーズになります。

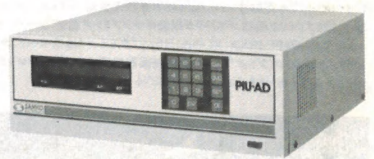
RS232Cをサポートしているパーソナルコンピュータとなら、頼もしいパートナーとなり得るインテリジェントタイプのI/Oユニットが、PIUシリーズです。パソコンの入出力装置として利用できるだけでなく、操作部、表示部を持っていますので、単独でも利用できます。機能面では、CPUを内蔵したインテリジェントタイプのためパソコンの負担を軽減でき、他にカレンダー付クロック、データのバックアップ機能なども備えています。また従来のように、パソコンと専用のインターフェイスによる構成システムでは、拡張性、耐ノイズ性、機能性などに問題がありましたが、PIUシリーズは多チャンネルでしかもすべての入出力がアイソレーションしてありますので、現状の不満も解消されるでしょう。お手持ちのパソコンとベストレーションとなるPIUシリーズを、どうぞお選び下さい。

## パソコン領域をぐんと広げる PIUシリーズ



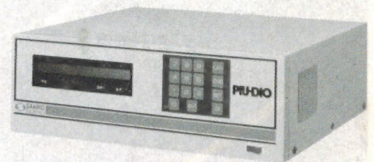
### PIU-AD

- アナログ入力96点、64点、32点、各タイプ
- 0~10V・±5V・1~5V・4~20mA
- フライングキャパシタ方式にて各入力は絶縁12BIT、A/D使用
- 内部クロックにより1分・10分・1時間の平均値と現在値データ有り
- スキャンングにて96点をA/D変換しており96点のループ時間は10秒・20秒・30秒の切換式



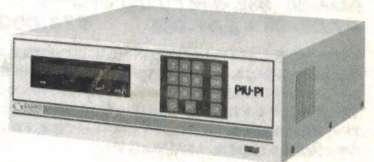
### PIU-DIO

- デジタル入力128点、出力128点
- 無電圧有接点入力、300mAオープンコレクタ出力
- フォトカプラにて絶縁
- 接点入力電源(+24V)は内蔵、オープンコレクタ用電源は内蔵せず



### PIU-PI

- パルス入力256点
- 無電圧有接点入力(ON30ms以下OFF40ms以上一チャタリング5ms以下)
- フォトカプラにて絶縁
- パルス数の積算カウント機能、カウンタは各チャンネル毎に0~99999まで指定CHのみカウントクリア可



### PIU-DA

- アナログ出力64点、32点
- 0~10V、±5V、1~5V
- フォトカプラにてデジタル側で絶縁シングルエンド
- 12BIT、D/A使用、キーボードより出力値の設定も可



## DIGITAL IC TESTER

## IC300

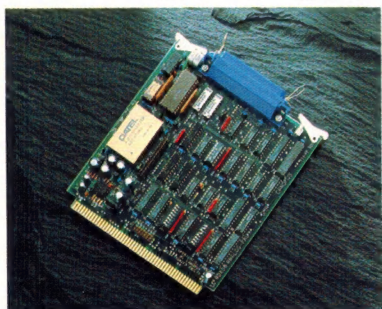


**三幸電子工業株式会社**

本社/〒468 名古屋市天白区天白町植田深田158 TEL (052) 805-2111代 FAX (052) 805-2500  
東郷工場/〒470-01 愛知県愛知郡東郷町春木白土1-84 TEL (052) 801-5251代 FAX (052) 801-5253

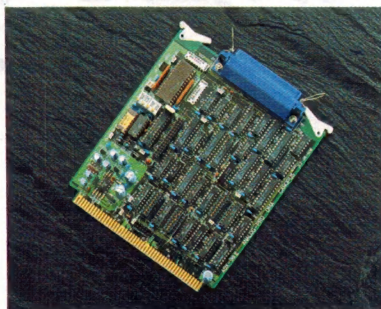


# 完璧なソフトサポート。即戦力として活用できるA/D、D/Aボード。



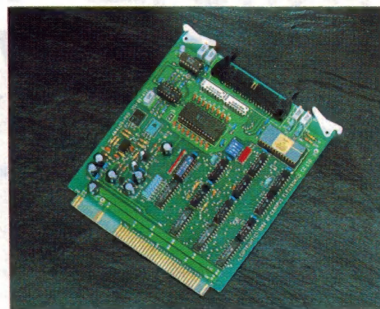
**DMAインターフェース付A/D変換ボード**  
● **ADX-98** ￥286,000

● 入力電圧範囲: -5V ~ +5V, -10V ~ +10V, 0V ~ +10Vから選択  
● 入力チャネル数: 8チャネル 差動/16チャネルシングル切替 ● 分解能: 12ビットバイナリ ● 直線性:  $\pm 0.03\%$ 以下 ● トリガ機能: TTLレベル入力端子によるアトリガ機能、ポストアトリガ機能 ● サンプルング周期:  $10\mu\text{S}$ /チャネル [高速モード(PC-98XA, XLのみ)]  $10\mu\text{S}$ /チャネル(通常モード) ● サンプルングクロック: プログラマブル水晶発振器内蔵または外部クロック ● デジタル入力: 4ビット ● デジタル出力: 4ビット



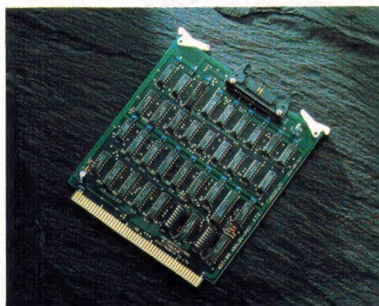
**高機能、使いやすくなった新ラインナップ**  
● **ADX-98E** ￥286,000

● 入力電圧範囲: -5V ~ +5V, -10V ~ +10V, 0V ~ +10Vから選択  
● 入力チャネル数: 8チャネル 差動/16チャネルシングル切替 ● 分解能: 12ビットバイナリ ● 直線性:  $\pm 0.03\%$ 以下 ● トリガ機能: TTLレベル入力端子によるアトリガ機能、ポストアトリガ機能 ● サンプルング周期:  $10\mu\text{S}$ /チャネル(通常モード)  $15\mu\text{S}$ /チャネル(低速モード) ● サンプルングクロック: プログラマブル水晶発振器内蔵、または外部クロック ● デジタル入力: 4ビット ● デジタル出力: 4ビット



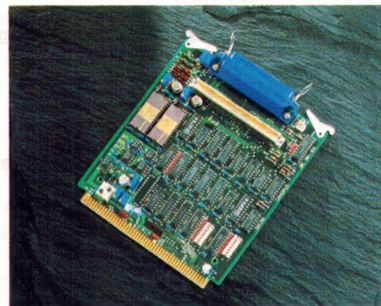
**12ビット、8チャネルA/D変換ボード**  
● **ANALOG-PRO I** ￥148,000  
● **ANALOG-PRO II** ￥168,000

● 入力電圧範囲: -5V ~ +5V, -10V ~ +10V, 0V ~ +10Vから選択  
● 入力チャネル: 8チャネル 各チャネルは差動入力 ● 分解能: 12ビットバイナリまたはオフセットバイナリ ● 直線性:  $\pm 1\text{LSB}$ 以下 ● 残留雑音:  $\pm 1\text{LSB}$ 以下 ● 変換時間:  $1/20\mu\text{S}$ (STYP),  $25\mu\text{S}$ (MAX) II/  $1/2\mu\text{S}$ (STYP),  $15\mu\text{S}$ (MAX) ● サンプルングクロック: プログラマブル水晶発振器内蔵、または外部クロック ● デジタル入力: 2ビット ● デジタル出力: 2ビット



**ADX-98 98E専用トリガコントローラ**  
● **ADT-98** ● **ADT-98E** ￥78,000

● トリガレベル指定: トリガレベル1, トリガレベル2は12ビットで指定。ソフトウェアのOUT命令により、1ビット×2回またはワード×1回 ● トリガデレイ指定: 0 ~ 65535クロック(ADX-98, 98Eのサンプルングクロック)。ソフトウェアのOUT命令で指定。1ビット×2回またはワード×1回 ● I/Oインターフェイス: I/Oアドレスは16ビットデコード。上位13ビットをデコードスイッチで設定。下位3ビットのすべてを使用 ● 消費電流: +5V, 700mA以下



**信号処理に使える待望16ビットD/Aボード**  
● **DAC-98** ￥158,000

● 入力コード: 16ビットオフセットバイナリまたは2Sコンプリメンタリバイナリ ● 出力電圧範囲: 0 ~ 5V, 0 ~ 10V, -5V ~ +5V, -10V ~ +10V ● 出力チャネル数: 2チャネル(オプションボードによりオンボードで最大8チャネル可能) ● 出力セトリグ時間: 5 $\mu\text{sec}$  ● I/Oアドレス: 16ビットデコード0または8から始まる連続8ビット使用(デバッグスイッチで選択) ● デジタル入出力: 各4ビットTTLレベル ● 消費電流: +5V 530mA, +12V 60mA, -12V 70mA(最大負荷時)



PC-9801シリーズ全機種  
(VX, VM21, XLを含む、  
LTを除く)に対応。

## ■A/D、D/Aボード型番

- ADX-98 ..... CAB-1104-11-14 × ×
- ADX-98E ..... CAB-1111-01-14 × ×
- ANALOG-PRO I ..... HAB-1101-11
- ANALOG-PRO II ..... HAB-1102-11
- ADT-98 ..... CAB-1105-11-14 × ×
- ADT-98E ..... CAB-1112-01-14 × ×
- DAC-98 ..... HAB-1110-01

## プログラマブルアナログ前(後)信号処理ユニット新登場。

# ASIP-0260

アナログ信号プリプロセッサ ￥198,000 型番HFU-7201-01H(L)

デジタル信号処理において要求されるアナログ信号前処理(後処理)機能をすべて含んだユニットです。カノープスのデジタル信号処理のノウハウをフルに活かして開発されました。

### 【ASIP-0260とは】

ASIP-0260はアナログ信号ラインとA/D入力またはD/A出力との間に接続するアナログ信号処理ユニットです。

プログラマブルゲインアンプ、アンチエイリアシング用プログラマブルローパスフィルタ、直流オフセット除去用ハイパスフィルタ、サンプルホールド回路、簡易LED出力モニタを備えています。

音声認識、音声合成、スペクトル分析など、ほとんどのデジタル信号処理に必要な不可欠なユニットです。

- 入力インピーダンス: 100K $\Omega$
- 振幅ゲイン : x1, x5, x10, x50
- ローパスフィルタ特性: 80dB/oct(6連チエビシェフ)  
<H型> fc(Hz) = 100, 200, 500, 1K, 2K, 5K, 10K, 20K  
<L型> fc(Hz) = 10, 20, 50, 100, 200, 500, 1K, 2K
- ハイパスフィルタ特性: 12dB/oct fc = 0.05(Hz)
- サンプルボード特性: アタックタイム 3.0 $\mu\text{sec}$ (0.01%)  
ドレアップ率 1mV/msec
- 出力インピーダンス: 50 $\Omega$ 以下
- 最大残留ノイズ : 0.2mV以下  
(条件 GAIN×50, 入力ショート, LPF OFF, HPF OFF)
- 最大チャネル間クロストーク: 1KHz -83dB以上
- 残留オフセット : HPF OFF  $\pm 3\text{mV}$ 以下  
HPF ON  $\pm 0.4\text{mV}$ 以下  
(GAIN×50入力ショート LPF ON)

# DIF-98

￥38,000 型番CFU-1102-01-14 × ×  
(サンプルソフト付)

### 【DIF-98とは】

ASIP-0260をコントロールするPC-9801シリーズ用デジタル出力インターフェースボードです。このボード1枚で2本のコントロールバスがドライブできますので、最大32台のASIP-0260を個別にコントロールできます。

- 使用アドレス: PC-9801のメモリー空間またはI/O空間のどちらかの連続した4バイトを使用

- ①メモリー空間使用時  
▶ 00000 ~ FFFFFFの任意の連続した4バイトをロータリースイッチとショートラグで指定  
▶ 出力専用なので9801内部のRAMエリアと重なっても使用可能(ただしプログラムで使用しないアドレス)

- ②I/O空間使用時  
▶ 0000 ~ FFFFの任意の連続した4バイトをロータリースイッチとショートラグで指定

- 拡張機能: ADX-98(E)カノープス製高性能A/D変換ボードのホールド信号により各チャネル同時サンプル1ホールドコントロール入力コネクタ(TTLレベル)を装備



# パーソナル・コンピュータ用FFTアナライザ/ デジタルオシロスコープソフトウェアの決定版。

## DSS 98

ADX-98(E), ADT-98(E), ANALOG-PRO I, II をフルサポート

**TYPE IV**  
(MS-DOS Ver2.0以上RAM容量512KB以上)

●PC-9801シリーズ用  
●PC-98XA/XL用 **各¥98,000**

型番SDP-1104-11-13 × ×  
(PC-98XA/XL用) 型番SDP-1205-11-13 × ×

### ■特長

- カノーブスのA/Dボードファミリーをフルサポート
- 512~8192ワードの可変フレーム長
- ブロック固定小数点演算による高速高精度FFT (PC-98XAの場合1024点で0.6秒)
- オシロモード・FFT1チャンネルモード・FFT2チャンネルモード・3次元表示モード・X-Y表示モード等豊富な表示モード
- 高速な波形・スペクトラム表示
- ウィンドウトリガ・アウトレンジトリガ・シングルショット等本格的なトリガ機能
- チャンネル間四則・微分積分・波形メモリ・アベレ

ーシング等強力かつ柔軟な演算機能

- Xカーソル・Yカーソルによる座標値やデータのデジタル表示
- MS-DOSファイルへのセーブ・ロード機能
- カーソル移動キーやファンクションキーによるユーザフレンドリーなオペレーション
- COPYキーによるプリンタへのハードコピー
- HELPキーによるヘルプメッセージ
- 仕様
  - 表示チャンネル数:最大8
  - フレーム長:512ワード~8192ワード
  - 表示項目:時間関数(波形),スペクトラム(パワー,実部,虚部,位相),ヒストグラム(振幅確率密度)
  - 表示モード:OSC 波形表示(8ch), X-Y X-Y表示(横軸1ch,縦軸7ch),X-YP X-Y表示(横軸4ch,縦軸4ch),FFT1 波形表示+スペクトラム表示(1ch),FFT2 スペクトラム表示(2ch),3D スペ

クトラム3次元表示(1ch) ●垂直軸:電圧軸(5V/div~0.01V/div),スペクトラム軸(2dB/div~80dB/div) ●水平軸:時間軸(250μsec/div~10sec/div),周波数軸(50KHz~1.25Hz) ●ズーム:時間軸,周波数軸と1/8~8倍のズーム ●垂直カーソル:X軸の2点の座標値と差を表示,波形の1周期検出機能,スペクトラムのピーク検出機能,電圧値・スペクトラムのレベルの読み出し ●水平カーソル:Y軸の2点の座標値と差を表示,波形の最大値と最小値検出機能,スペクトラムのピーク検出機能 ●オーバーラップ表示:任意のチャンネルを重ね書き ●ハードコピー:COPYキーによる画面のハードコピー ●サンプリング点数:512点~8192点(フレーム長に依存) ●サンプリング周波数:100KHz~2.5Hz(時間軸に連動) ●FFT点数:512点~8192点 ●周波数分解能:サンプリング周波数/FFT点数 ●ウィンドウ関数:Dirichlet(短形窓)Blakman(ブラックマン窓)Fejer(三角窓) Gaussian(ガウシアン窓) Hanning(ハニング窓) User(ユーザー定義窓) Hamming(ハミング窓) ●ヒストグラム点数:256点 ●アベレージングモード:リニア加算平均,指数加算平均(3種),ピークホールド(スペクトラムのみ) ●アベレージング回数:1~32000回の任意 ●演算:四則,微分,積分 ●メモリ機能:任意のチャンネルのデータを保持

## 信号処理の夢、ここに実現。

【SAL】は、カノーブスが独自に開発した信号処理用コマンドを備えた科学技術計算用言語です。インタープリタの使い良さとコンパイラと速度を兼ね備え、いかなる言語よりも使いやすく洗練されています。

# SAL

信号処理用コマンドを備えた科学技術計算用言語

●PC-98XA/XLバージョン

**¥480,000**

型番SDP-1206-01-13 × ×

### ■信号処理コマンド

SALの信号処理用コマンドは充実しています。9種の内蔵ウィンドウ関数から、FFT、コヒーレンス、伝達関数まで、関数電卓のキーを押す感覚で使えます。

### ■演算コマンド

配列と変数、配列と配列の演算が1つのコマンドで、しかも高速に行なえます。したがってインタープリタでありながらコンパイラなみの実行速度が得られます。また、単精度整数型から倍精度実数型まで4種のデータタイプと、複素数型をサポート。関数は常用ものを網羅。もちろんこれらにもアレイ演算機能が適用できます。

### ■A/D変換、D/A変換コマンド

A/D変換器、D/A変換器には定評あるADX-98(E),

ADT-98(E)DAC-98を採用。10μS/chの高速変換とアナログ信号によるトリガを実現しています。A/D変換して得られたデータを加工し、D/A変換し出力することもSALを使用すれば簡単にこなせます。

### ■グラフィックスコマンド

データ描画用のコマンドにはSCALE(罫線描画)とDRAW(データ描画)を用意しました。これによってデータの種類によらず罫線やデータが描画されます。また基本グラフィックスについてはN88-BASIC相等のものにパラメータに配列が使える形で拡張しました。このため1回の命令の実行で複数の直線や円を高速描画できます。さらに10個までのプレーンを定義でき、それぞれのプレーンに独立して描画できます。

### ■プログラミング

各コマンドがマクロ的な動作をしますので、パラメータやコマンドを順に並べるだけで不必要なループもなく目的の処理が行なえます。そのうえ一連のコマンドをマクロ命令(サブルーチン)として定義し、新たなコマンドとして使用できます。もちろんプログラミング言語としての機能は、ループや条件分岐を含めてすべて持っていますので、より高度なプログラムも作成できます。

### ■そのほかにも

- オンラインヘルプを装備
- 5つのマルチキャラクター・スクリーン
- 70文字×20キーのプログラマブル・ファンクションキー

## ANALOG-PROシリーズをサポートするハンドラソフト。

**ANALOG-PRO** ハンドラソフト

(N88-BASIC用、C用、FORTRAN用含む) [ニューバージョン]

**CHS-ADT-606 ¥10,000**

型番SAB-1108-11-13 × ×

**ハンドラソフト用FFTルーチン**

(CHS-ADT-606に組み込み使用)

**CHS-FFT-606 ¥30,000**

型番SAB-1109-11-13 × ×

●それぞれ、MS-DOS用ソース(MASM)・サンプルプログラムと、DISK-BASIC用オブジェクト・サンプルプログラムのセットです。●ディスクサイズ(82D:5.2DD:5.2HD:3.52DD)をご指定ください。

■N88-BASIC、C、FORTRANから簡単に使えます。

■アナログシリーズ全機種サポート

■トリガ機能も充実。

■代表的なC・FORTRANをサポート

■全ソースを公開

■目的に応じて使える4つのモード。

○スモール・メモリ・モード

64Kバイト以内のデータを高速に記録するモードです。

○バンク・メモリ・モード

バンク切り換え方式のメモリに大量のデータを連続

して記録します。(最大32MB(9801)-15MB(XA))

○ラージ・メモリ・モード

64KB以上のデータを記録できます。

○インタラプ・モード

並行動作が可能です。(ADX-98では改造が必要)

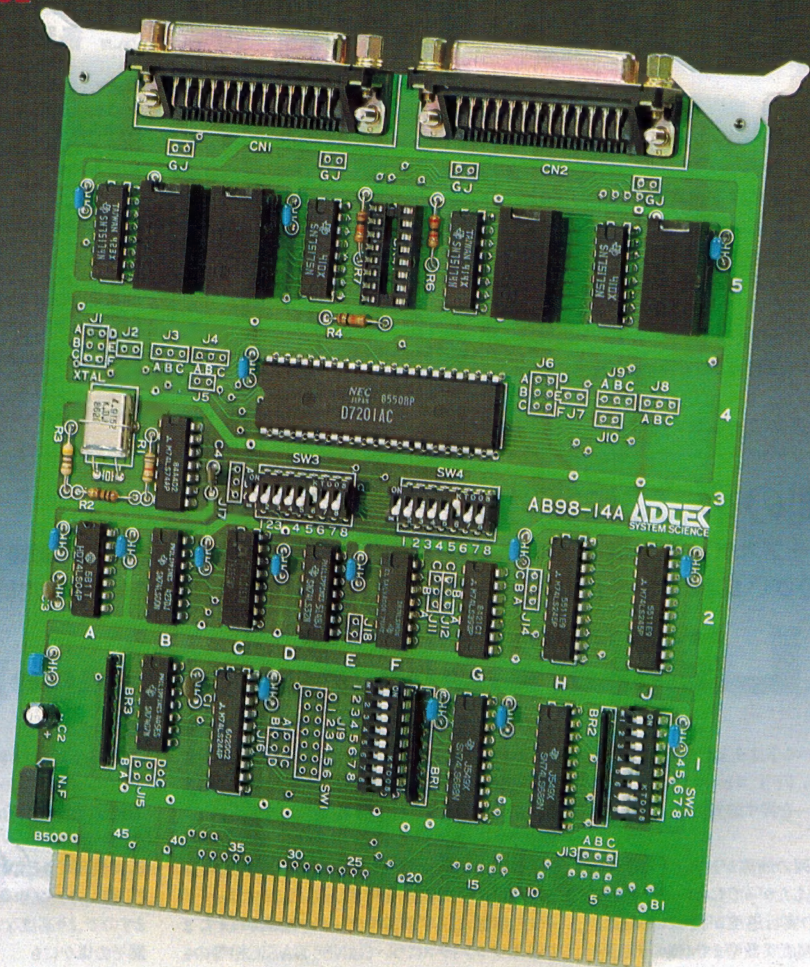
■ご注文時には型番をご指定ください。型番内の××はディスクのサイズを示します。/82D:8D.52DD:5W.52HD:5H.3.52DD:3W.3.52HD:3H

# CANOPUS

カノーブス電子(株)

本社〒658 神戸市東灘区西岡本1丁目4-30カノーブスビル ●Phone:078-411-5292(代)  
<テクニカルインフォメーション>078-412-7166(月曜~金曜/PM1:00~3:00)





## 高性能、高信頼性ボード。AB98シリーズ。 PC-9801シリーズ用拡張インターフェースボード群。

高性能、高信頼性を誇るAB98シリーズは、NEC PC-9801シリーズの应用能力を一段と向上させ、多彩なニーズにお応えできるインターフェースボードファミリです。豊富なラインナップはもとより、この優れた拡張性は、特にLA・FAの分野でのニーズに最適です。

- AB98-01 ユニバーサルボード…………… ¥ 4,200
- AB98-02A 非同期式シリアルインターフェースボード…………… ¥ 45,000
- AB98-03A PROMライタボード…………… ¥ 40,000
- AB98-04A 48ビットパラレルI/Oボード…………… ¥ 36,000
- AB98-05A 8ch12ビットA/Dコンバータボード…………… ¥ 68,000

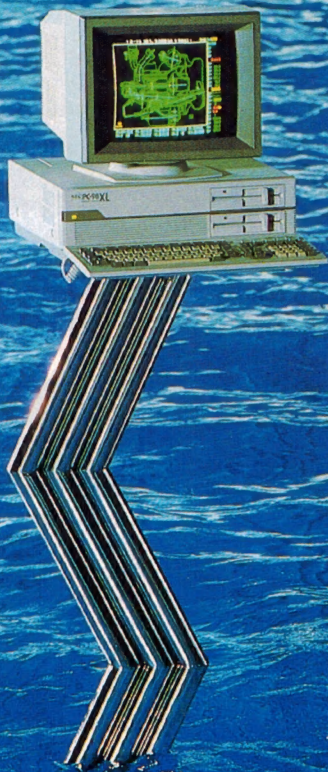
- AB98-06A 2ch12ビットD/Aコンバータボード  
V(電圧出力)…¥ 59,500 / I(電流出力)…¥ 69,500
- AB98-07A ステッピングモータコントローラボード…………… ¥ 68,000
- AB98-08A 24chリレーボード…………… ¥ 55,000
- AB98-09A 24chアイソレーション入力バッファボード…………… ¥ 35,000
- AB98-10A 絶縁型8ch 入・出力ボード…………… ¥ 35,000
- AB98-11A 絶縁型8/16ch MOS FET出力ボード…………… ¥ 34,000
- AB98-12A 4ch交流リレーボード…………… ¥ 29,000
- AB98-13A 4ch直流リレーボード…………… ¥ 49,500
- AB98-14A 2ch RS422インターフェースボード…………… ¥ 57,000



つながるぞ ひろがるぞ NECのパソコン

上はかな性で世界をもつ  
**NEC**

# 98最上級。



PC-9800シリーズの最上位機PC-98XLは  
98シリーズの豊富にそろったハード/ソフト資産を幅広く包含。  
さらに進化した最先端の機能で  
プロフェッショナルの高度なニーズに応えます。

●1,120×750ドットの高速・超高解像度グラフィックス ●ハイレゾリューション  
モード(1,120×750ドット)とノーマルモード(640×400ドット)の2つのグラ  
フィックモードをサポート ●新世代CPU80286(8/10MHz)と、μPD70116-10  
(V30)(8/10MHz)を搭載 ●大容量1Mバイトユーザーズメモリを標準実装  
(最大7.5Mバイト・80286CPU使用時) ●PC-9801VM/VXからPC-98XAまでの  
PC-9800シリーズの膨大な資産を継承 ●用途に応じて選べる3モデルラインナップ

NECパーソナルコンピュータ  
PC-9800シリーズ

## PC-98XL

model 1 1Mバイトタイプフロッピーディスクインタフェース内蔵……………本体標準価格 495,000円  
model 2 1Mバイトタイプ5インチFDD2台内蔵(写真)……………本体標準価格 575,000円  
model 4 1Mバイトタイプ5インチFDD2台、20Mバイトタイプ3.5インチ固定ディスク1台内蔵……………本体標準価格 835,000円



トップシェアの実力、さらに進化。  
新・PC-9800シリーズ



PC-98LT ● PC-9801UV2 ● PC-9801VM21 ● PC-9801VX0/VX2/VX4 ● PC-98XL

日本電気グループ

■お問い合わせは、最寄りのNECへ

北海道支社(札幌)011(251)5531/東北支社(仙台)022(261)5511/東京支社(東京)03(456)3111  
中部支社(名古屋)052(262)3611/北陸支社(金沢)0762(23)1621/関西支社(大阪)06(231)3111  
中国支社(広島)082(247)4111/四国支社(高松)0878(22)4141/九州支社(福岡)092(271)7700

■技術的なご質問・ご相談に電話でお答えします。

NEC パソコンインフォメーションセンター

東京 03(452)8000 大阪 06(211)9800

受付時間 9:00~17:00 月曜日~金曜日(祝日を除く)

(電話番号は、よくお確かめの上おかけください)

**C&C**  
コンピュータ・アンド・コミュニケーション

NECのパソコンファミリー





DECnet-DOS/9800  
ローカルエリアネットワーク

NECパーソナルコンピュータ  
PC-9801シリーズ用

# DEC社製VAXシリーズを ETHERNETでサポート。

## CSMA/CD LANモジュール

### FA-LANII(98)D

FA-LANII(98)Dは、PC-9801シリーズパソコンとDEC社製VAXシリーズ、PDP-11シリーズをETHERNETで結び、DECnetを構築することができるCSMA/CD LANモジュールで、大量のデータを高速かつ正確に伝送します。サポートソフトウェアとして、DEC社製DECnet-9800が豊富な機能を提供いたします。

#### ●仮想端末機能

PC-9801側からコマンドひとつでVAXに対してリモート・ログインすることができます。端末はもちろんVT100フルスクリーン・エミュレーション。VAXのローカル・ターミナルのように動作します。

#### ●ファイル転送機能

PC-9801側のディスク装置とVAXに接続されている大容量ディスク装置間で、双方向にデータファイルの転送が可能です。MS-DOS順成ファイルのテキストデータはもちろん、バイナリデータも可能です。

#### ●仮想ディスク機能

PC-9801側のユーザは、VAXの大容量ディスク装置をローカル・ディスクのように利用することができます。

#### ●仮想プリンタ機能

PC-9801側から、リモートのVAXに接続された高速プリンタをローカル・プリンタのように利用することができます。

#### ●電子郵便サービス

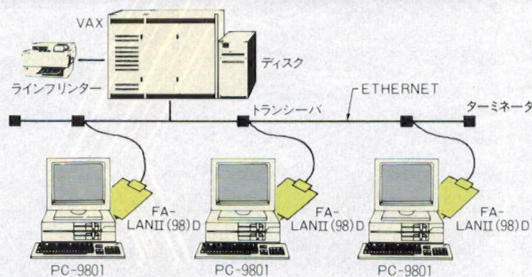
VAX Mailの機能を使うことによって、パソコン・ユーザはVAX上に自分自身の私書箱を設定できます。

#### ●電子掲示板/電子会議システムサービス

VAX Notesの利用によって、パソコン・ユーザが情報交換のためのサービスを利用することができます。

#### ●ネットワーク管理機能

ネットワーク上の各種の統計情報やエラー情報の監視と制御を行うことができます。



- VAX, PDP, DECnet, VT100, VAX Mail, VAX Notesは米国DEC社の製品です。
- ETHERNETはXEROX社の登録商標です。

#### ■仕様

伝 送 速 度	10Mビット/秒
フレーム間スペース時間	9.6 μs
ス ロ ッ ト 時 間	51.2 μs
伝 送 バ ン ド 方 式	ベースバンド方式
伝 送 コ ー ド	マンチェスタ・コード
最 少 フ レ ー ム 長	64バイト
C R C の 形 式	32ビットAutodin II CRC多項式
ス テ ー シ ョ ン 台 数	最大1024(リピータ使用時)
トランスシーバケーブル長	最大50m 4対ツイストペア・ケーブル
同 軸 ケ ー ブ ル 長	最大2500m 50 Ω 同軸ケーブル(リピータ使用時)
ス テ ー シ ョ ン 番 号	コマンドにて6バイト長を設定
使 用 C P U	i82586(インテル社製)

上記仕様は、IEEE802.3規格に準拠し、かつETHERNETの規格にコンパチブルなデータ転送が行えます。

#### セミナー & スクール のご案内

- iRMX86(98)無料セミナー
- iRMX86(98)オペレーション入門コース
- PC-MODULE実習コース
- MODULE-PAC(98)無料セミナー
- LA-PAC(98)無料セミナー

《お問合わせ先》●(株)理経●(株)住商エレクトロニクス●(株)東京エレクトロ●三菱事務機械(株)  
●(株)ミウラ・広島●九州電子機器サービス(株)●全国のNECマイコンショップ

FA&OA  
CONTEC  
株式会社コンテック

コンテック/FAマイコンセンター  
〒105 東京都港区芝2-29-11  
TEL (03)769-1061 FAX (03)769-1060  
NEC マイコンショップ  
コンテック マイコンセンター  
〒555 大阪市西淀川区堀里3-9-31  
TEL (06)472-0265 FAX (06)475-1728



# CS<sub>dc</sub>

標準品／特注標準品／特注品

# シリコン高圧整流素子

CSdc社は、高信頼度を誇るシリコン高圧整流素子を供給しています。その多種多様な標準品により、どのような目的にも最適な機種を選択して頂くことができます。

また、用途により特別な仕様に対しては、カスタムメイドにも応じられます。

〈用途〉 レーダー、X線装置、CRT モニター、レーザー装置、静電集塵装置、高圧安定化電源 その他。

## 小型高圧整流ダイオード

### NKVシリーズ

■3KV、70mA ～ 20KV、5mAまで6機種。■200nano Sec。

■-40℃～125℃

### JKVシリーズ

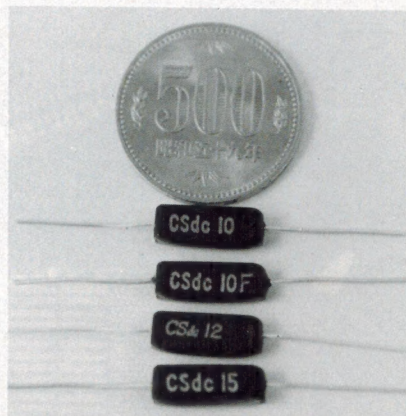
■8KV、220mA ～ 15KV、60mAまで8機種。■200nano Secの高速スイッチングタイプあり。■-40℃～125℃。

### Hi-Slimシリーズ

■50/60Hzタイプ：2.5KV、2A～25KV、500mAまで13機種。

■250nano Secタイプ：2.5KV、500mA～15KV、1.4Aまで11機種。

■-55℃～125℃(50/60Hz)・-55℃～110℃(250nano Sec)



— JKVシリーズ —

## “IR”シリーズ高圧整流素子

■International Rectifier 社の旧製品に1:1で対応する。■～75KV、～1.25Aまで多機種。

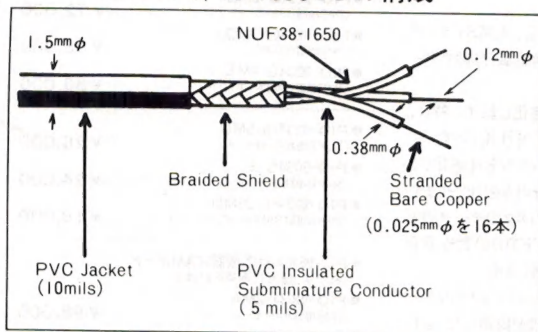
■高高度でのフラッシュオーバーとコロナ放電を最少化している。

■モールドスティック型には半波整流と両波整流の2方式がある。■-55℃～140℃。

☆特注による高圧安定電源装置 承ります。

## COONER社製 WIRES & CABLES

### NMUF3/38-1650SJの構成



### 電磁ヘッド用多芯シールド線

### ミニ電磁ヘッドケーブル(MGHシリーズ)

コンピュータ用の各種電磁ヘッドの多線接続用に開発された細く柔軟性に富んだ多芯シールドケーブルである。3芯から11芯まで数多くのモデルがある。米国の主要メーカーに納入され、高い評価を受けている。

### 極細リード線

### ミニフレックス・ワイヤー(NUFシリーズ)

超柔軟性の極細銅線(0.025mmφ)を16～65本で撚りあげた0.12～0.30mmφの導体を0.127mm厚の塩化ビニールで絶縁。温度範囲-55℃～+90℃、電圧定格200VDC。外被カラー10色。

### 極細多芯シールド線

### ミニフレックス・ケーブル(NMUFシリーズ)

芯線としてNUFシリーズ極細リード線を使用。標準芯線数は1～5本。外側仕上げはシールド及び塩ビジャケット。

取扱店：

(株)クローネ

[03]695-5431

スワロー電子(株)

[044]945-0345

CSds 社総代理店

**ゼネックス** 株式会社

106 東京都港区西麻布1-14-15  
TEL (03)470-3970  
FAX (03)470-5104

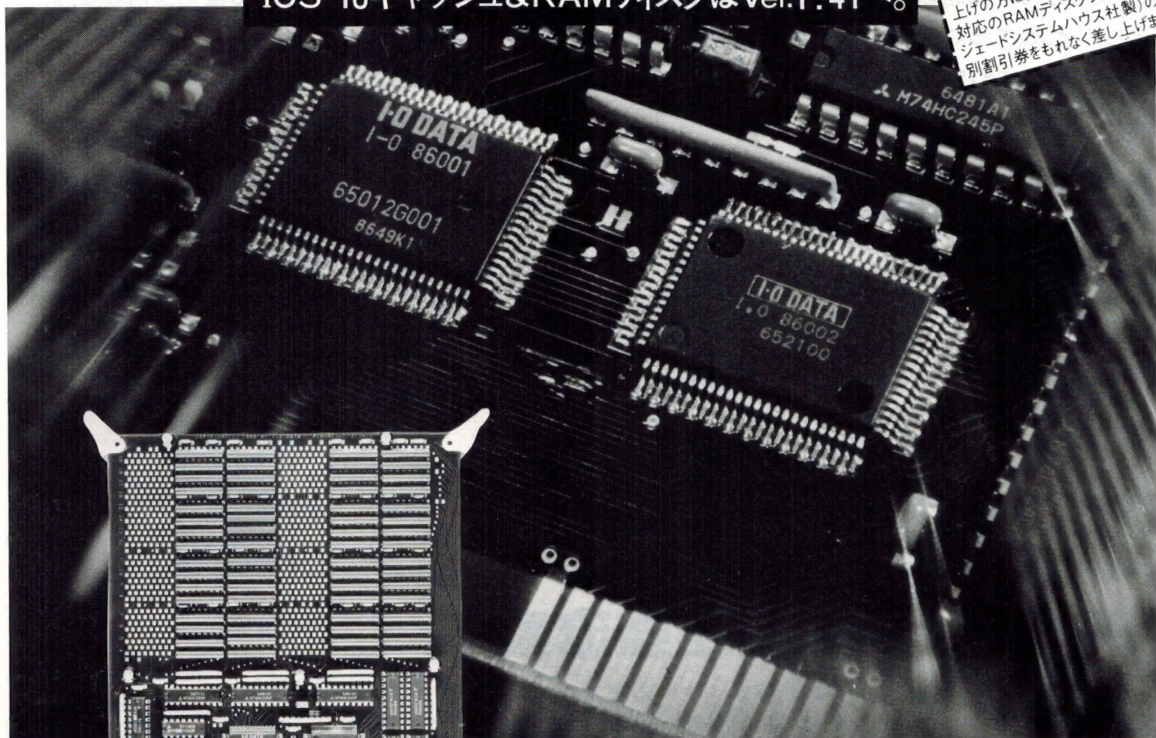


# 夢のHiSUM機能\*を標準搭載。(新登場)

Gシリーズは、すでに9234シリーズをご利用のお客様にも効果を発揮します。

**IOS-10キャッシュ&RAMディスクはVer.1.41へ。**

1MB以上の9234Gシリーズをお買い上げの方に、**Non-DISK BASIC(86)** 対応のRAMディスクソフト("JSC" 対応システムハウス社製)の特  
別割引券をもれなく差し上げます。



※HiSUM機能とは……(特許出願済)

High Speed Sum Check Circuitの略で、パソコンのバンク増設RAMに必要なデータチェック方式で、メモリー間のデータ転送時にデータバスを同時監視チェックするハードウェア回路です。これにより98用メモリー装置に不可欠な確実チェックができ、しかも他のI-Oバンクメモリーボードにも有効に働きます。ソフトウェアチェックでは実現できない約2倍(当社比)の高速転送が実現する独創機能です。

(HiSUM回路は、他社のRAMディスクソフトでは、サポートされていないので、全くスピード・アップしません。ご注意ください。又、本機能をサポートされるソフトハウス様には詳細資料をお送りします。

## 新設計、9234Gシリーズ (カスタムICを2個搭載)

- HiSUM®回路(高速データチェック回路)をカスタムIC化して、RAMボードに組み込みました。従来、ソフトウェアでデータのサムチェックを行っており、速度が抑えられておりましたが、Gシリーズでは、ハードウェアが代行しますので夢の高速かつ高信頼が実現しました。
- 低消費電力設計。周辺回路をC-MOSタイプカスタムIC化し、さらに、消費電流を小さくしました。
- HiSUM回路は9234GシリーズRAMの全てに標準搭載。GシリーズRAMは、従来の9234シリーズおよびI-Oバンク®方式の他社RAMボードとの混在利用が可能です。したがって、1枚のGシリーズRAMが高速レスポンスの利用環境を実現します。

### お知らせ

PIO-9234シリーズ用のRAMディスクの旧バージョンをご使用の方はIOS-10V1.41へバージョンアップします。MX-1plusからは¥5,000、IOS-10からは¥2,000です。シリアルNo.、バージョンNo.、希望メディアタイプを記入して本社に直接お申込み下さい。

●32MBまでサポートしたRAMディスク(IOS-10-32M)もあります。¥10,000仕様：汎用RAMディスクタイプでMAX32MBをサポート。(標準IOS-10は、16MBまで) 制限事項：NECの辞書は使えません。

## IOS-10 Ver.1.41キャッシュ&RAMディスク

- EMS方式のロータス1-2-3にも対応。(1-2-3)のワークシート領域を、バンクRAMにも拡大できます。
- IOS-10では、EMS方式対応にも、データの信頼性を最重視し、確実なデータチェックを行なっています。
- RAMディスク、ディスクキャッシュ、EMSドライバの全てに、HiSUM回路の自動認識、自動サポート機能を組み込みました。
- ディスクキャッシュをさらに、高速化しました。特に、書き込み時の速度は、2倍以上(当社比)です。
- D COPY、BAT COPYユーティリティを高速化しました。FATをサーチし、必要部分のみをコピーします。
- 1メガバイトタイプフロッピーディスクのモータのステータスを3msに設定し、8" FDDの動作音を静かにするユーティリティが付属します。
- IOS-10ディスクキャッシュは、ハードディスクのユーザにも小気味よいレスポンス環境を提供いたします。

▶IOS-10は、5"-2DD規格を添付しておりますが、PC-9801VM2/VXでは2HDに交換して使用できます。

## Gシリーズ増設RAMボード

(IOS-10 Ver. 1.41付き)新発売!

●PIO-9234G-1MD (1MB増設RAMボード) .....	¥ 45,000
●PIO-9234G-1.5MD (1.5MB増設RAMボード) .....	¥ 58,500
●PIO-9234G-2MD (2MB増設RAMボード) .....	¥ 72,000
●PIO-9234G-3MD (3MB増設RAMボード) .....	¥ 85,000
●PIO-9234G-4ME (4MB増設RAMボード) .....	¥ 89,000
●PIO-9234-0.5MD (512KB増設RAMボード) .....	¥ 26,000
●PIO-9034C-3 (384KB増設RAMボード) .....	¥ 24,000
●PIO-9234-0.25MD (256KB増設RAMボード) .....	¥ 18,000

\* PC-98XA対応増設RAMボード  
(XLでは、使用上、制限があります)

●PIO-9X34-2MA  
(2MB増設RAMボード) .....

●ロータス1-2-3は、Loutous社の商標です。

情報と制御のシステムメーカー

**株式会社 I-Oデータ機器**

本社営業部：〒920 石川県金沢市駅西本町1-5-41  
東京営業所：〒101 東京都千代田区岩本町3-3-14(OMビル3F)

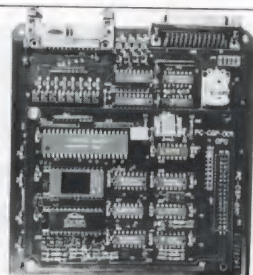
TEL. (0762) 21-4812 FAX. (0762) 63-0024  
TEL. (03) 866-3583 FAX. (03) 851-7160



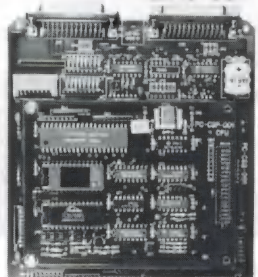
# インテリジェント インターフェイスボード

PC9801に.....

## マルチプロセッサ機能を！



▲PC-CSP



▲PC-CSS

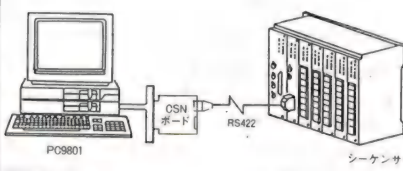
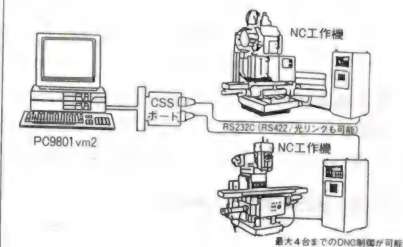
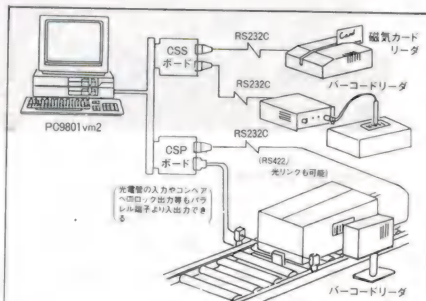
PC-9801シリーズのFAシステムへの応用の拡大にともない、バーコードリーダ、カードリーダ、シーケンサ、各種制御盤など、多数の入出力機器とシリアル通信やパラレル入出力で接続の必要性が増大しています。

これに伴って、「プログラムは使い易いN88BASICでやりたいが、シリアルボードが少なく制御が複雑で実行時間がかかりすぎる」「増設ボードをBASICで制御すると応答が遅すぎる」等の問題が指摘されています。

PC-CSS及びPC-CSP、PC-CSNは、独自のCPUを持つインタフェースボードです。PC-9801の拡張用スロットに挿入して使用することによって、多数の周辺機器との接続が可能となり、応答速度も飛躍的に改善されます。同時にPC-9801のI/O制御の負担を軽減させ、その能力を2倍以上引き上げることができます。

### 【特徴】

- (1) Z80 CPU、プログラムROM、電池バックアップSRAMを内蔵し、独自にI/Oを制御。
  - (2) 標準の「I/O制御ROM」と「PC-9801用専用サブルーチン」(機械語)を使ってインテリジェントインターフェースとなります。また各種周辺機器に応じた専用プログラムROMにより、より効果的な利用が可能。
  - (3) 電源バックアップ付SRAM(最大8Kバイト)は小規模外部メモリとして使用可能。
  - (4) 最大4枚までのボードを、同時使用できる。
- ※専用プログラム開発などはお問い合わせください。



### PC-CSS

RS232Cポート×2

バーコードリーダ、磁気カードリーダ等の各種I/O機器の制御をボード上のCPUが独自に行なう。そのためPC9801はほとんど負担なくRS232Cポートを増設できる。最大4枚、8ポートまで増設可。

### PC-CSP

RS232Cポート、パラレルI/O

RS232Cポートに付け加え、フォトスイッチやリミットスイッチなどの入力を直接とりこめるパラレル入出力をもっています。専用プログラムを作成すると、コンベア制御装置やロボット制御装置として利用できます。

### PC-CSN

RS422/マルチドロップ、RS232C

RS422/マルチドロップによる小規模のネットワークを構成できます。パソコン相互間のネットワーク、シーケンサとのネットワークなどを容易に構成できます。

## ローカル・インディケータ BLDS-LT



BLDS-LTはパソコンやミニコンの端末装置として、FAシステムの作業指示装置として開発された機器組込み型の、多桁表示装置です。

### 【特徴】

- (1) 大型で明るく見やすい32桁アルファニューメリック表示。
- (2) 外部ランプ表示、シーケンサと直結できる16ビット出力。
- (3) スイッチ、キーボード入力を取り込む8ビット入力。
- (4) シリアル通信で転送されたデータの表示、最大63の表示パターンの登録と任意のパターンの画面表示ができる。
- (5) パラレル入出力の通信による制御が可能。

### NC工作機用自動プログラミングソフトウェア

MA PLE-NC98  
MA PLE-PC98  
MA PLE-AT98

当社は2軸、2½軸、3軸加工をサポートする各種自動プログラミングソフトウェアを開発しています。

**BOLC**

有限会社 ボルク電子

〒543 大阪市天王寺区大道3-7-8  
☎(06) 773-2932 FAX (06) 779-8828



新・発売

# EPLライター

MODEL PW98-20

(株リコー認定品)

消去・再書き込み可能なPLD=EPLの書き込み用ボードです。

PC-9801シリーズの増設用スロットに入れて使用します。

EPL専用化により低価格を実現しました。

使いやすい書き込みソフトを用意しました。

## ■PW98-20仕様

使用パソコン NEC PC-9801シリーズ  
(但し、PC-9801XAを除く)

使用法 増設用スロットに装着

書き込み可能デバイス (株リコー製EPL (EPL10P8A/B, EPL12P6A/B, EPL14P4A/B, EPL16P2A/B, EPL16P8B, EPL16RP8B, EPL16RP6B, EPL16RP4B))

書き込みソフト EPLW.EXE (MS-DOS)

供給メディア 5インチ2DD又は、5インチ2HD

オプション EX-20 延長用書き込みソケット(ケーブル付き)

## ■読み込み可能なファイル

ABEL (データI/O社) JEDECファイル

EPLASM (株リコー) JEDECファイル

PALASMI (MMI社) HEXファイル

PALASMI II (MMI社) JEDECファイル

●上記各社のファイルの内、置換可能なデバイスのファイルを読み込み、そのデータを対応するEPLデバイス用に変換します。

●論理式をJEDECファイルに変換するソフトとしては、ABEL(データI/O社)、EPLASM(株リコー)、PALASMI I,II (MMI社)等のソフトを別に御用意下さい。

## ■定価

PW98-20 ボード(書き込みソフト付き) 98,000円

EX-20 延長用書き込みソケット(ケーブル付き) 8,000円

## ■EPLと置き換え可能なPALのリスト

PAL10L8	PAL10H8	▶EPL10P8A/B
PAL12L6	PAL12H6	▶EPL12P6A/B
PAL14L4	PAL14H4	▶EPL14P4A/B
PAL16L2	PAL16H2	PAL16C1 ▶EPL16P2A/B
PAL16L8	PAL16P8	▶EPL16P8B
PAL16R8	PAL16RP8	▶EPL16RP8B
PAL16R6	PAL16RP6	▶EPL16RP6B
PAL16R4	PAL16RP4	▶EPL16RP4B

株リコー製EPLはMMI社、NS社の上記のPALとは機能的にピンコンパチブルです。

●EPLを使用する為の入門の資料をお送り致します。EPLに興味をお持ちの方、これからEPLを使ってみたいとお考えの方はお申し込み下さい。(無料)

●EPLを使用した回路の設計、試作のお手伝いを致します。(有料)

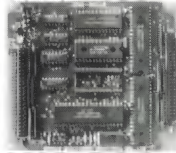
**東京チューナー工業株式会社**

東京都杉並区堀之内1-5-5 〒166 TEL.03(311)6631



# 充実のラインナップ Z80ワンボードコンピュータ & 周辺インターフェイス

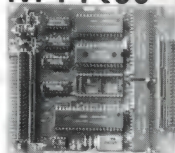
## MYK80-16KB ワンボードコンピュータ



¥18,800

メモリーバックアップ可能  
リセットIC採用  
ICソケットは全て丸ピンソケット  
Z80A CPU/2764ROM,JPにて27128、27256可/6  
264RAM/8255×2/Z80ACTC/TL7705リセットIC/  
115×130mm(基板寸法)/50PFC用コネクタ×2/4  
MHzクロック

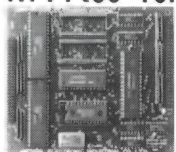
## MYK80-16PB ワンボードコンピュータ



¥18,800

メモリーバックアップ可能  
リセットIC採用  
ICソケットは全て丸ピンソケット  
Z80A CPU/2764ROM,JPにて27128、27256可/6  
264RAM/Z80APIO×2/Z80ACTC/TL7705リセ  
ットIC/115×130mm(基板寸法)/50PFC用コネクタ  
×2/4MHzクロック

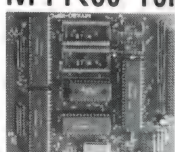
## MYK80-16KC ALL CMOS ワンボードコンピュータ



¥23,000

CPU.....LH5080L(Z80CPU CMOS)  
ROM.....2764×2(2732,27128,27256可)丸ピンソケット  
RAM.....6116(6264可)  
IO.....82C55×2  
CTC.....LH5082L(Z80CTC CMOS)  
基板寸法.....115×130mm  
コネクタ.....50PFC用×2  
クロック.....2MHz CMOSタイプ  
ゲート.....74HC シリーズ  
消費電流.....25mA type.

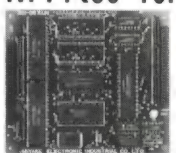
## MYK80-16PC ALL CMOS ワンボードコンピュータ



¥23,000

CPU.....LH5080L(Z80CPU CMOS)  
ROM.....2764×2(2732,27128,27256可)丸ピンソケット  
RAM.....6116(6264可)  
IO.....LH5081L(Z80 PIO CMOS)×2  
CTC.....LH5082L(Z80 CTC CMOS)  
基板寸法.....115×130mm  
コネクタ.....50PFC用×2  
クロック.....2MHz CMOSタイプ  
ゲート.....74HCシリーズ  
消費電流.....25mA type.

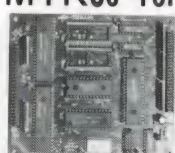
## MYK80-16K ワンボードコンピュータ



¥18,800

RAM 8KB  
ROMソケット 丸ピンソケット  
Z80ACPU/2764ROM,JPにて2732,27128,27256  
可/6264RAM/8255×2/Z80ACTC/115×130mm  
(基板寸法)/50PFC用コネクタ×2/4MHzクロック

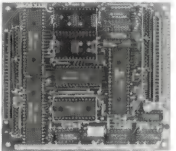
## MYK80-16P ワンボードコンピュータ



¥18,800

RAM 8KB  
ROMソケット 丸ピンソケット  
Z80ACPU/2764ROM,JPにて2732,27128,27256  
可/6264RAM/Z80APIO×2/Z80ACTC/115×  
130mm(基板寸法)/50PFC用コネクタ×2/4MHz  
クロック

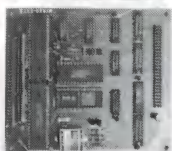
## MYK80-8K ワンボードコンピュータ



¥18,000

Z80ACPU/2732ROM  
×2/6116RAM/8255×  
2/Z80ACTC/115×13  
0mm(基板寸法)/50PF  
C用コネクタ×2/4M  
Hzクロック

## MYK80-EXT II メモリー I/O 拡張

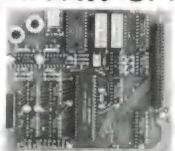


¥18,000

6264 8KB MAX16KB  
(RAM) バッテリーバック  
アップ付 / GB50-3(バッ  
テリー)/S-8054ALR(停  
電検出)/8255×2/115  
×130mm(基板寸法)/50  
PFC用コネクタ×2

新製品

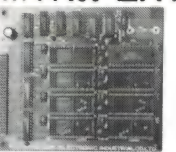
## MYK80-SI シリアル インターフェイス



¥18,000

Z80ASI O/O  
MC14411P(ボーレート)  
RS232C 2チャンネル  
5V単一電源(±12V内  
蔵)  
115×130mm(基板寸法)  
/バス50PFC用、232C  
20PFC用コネクタ

## MYK80-EXTM メモリー拡張



¥18,000

メモリー32KB、2バンクま  
で拡張可能  
ROM2764、RAM6264  
をジャンパーにて設定、  
6264は16KB、16PBに  
てバッテリーバックアップ  
可能  
標準6264、1ヶ実装  
基板寸法115×130mm

## MYK80-PPI I/O拡張



¥18,000

8255×4  
12ポート96ビット  
入出力拡張  
50PFC用コネクタ×3  
基板寸法115×130mm

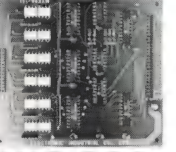
## MYK80-PIO I/O拡張



¥18,000

Z80APIO×4  
8ポート64ビット  
入出力拡張  
50PFC用コネクタ×3  
基板寸法115×130mm

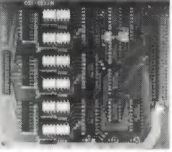
## MYK80-ISI アイソレーション 入力



¥18,000

24bit入力  
TLP521(フォトカブラ)  
バス50PFC用、入力26  
PFC用コネクタ  
115×130mm(基板寸法)

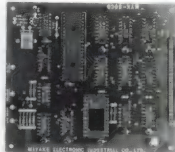
## MYK80-ISO アイソレーション 出力



¥18,000

24bit出力  
ULN2803Aオープンコレ  
クタ  
TLP521(フォトカブラ)  
バス50PFC用、出力26  
PFC用コネクタ  
115×130mm(基板寸法)

## MYK80-CD キャラクタ ディスプレイ



¥15,000

HD46505S(CRTC)  
2114×2(RAM)  
2716(CGROM)  
40×25文字以内表示  
115×130mm(基板寸法)

## MYK80-UNI MYK80シリーズ用ユニバーサル基板 (シリーズ同寸法、ガラス基板).....¥3,000

近日発売 MYK180シリーズ  
Z80上位コンパチブル64180  
ワンボードコンピュータ周辺インターフェイス

ミヤケ電子工業株式会社  
〒607 京都市山科区勸修寺福岡町19-3  
TEL.075(501)2022 FAX.075(591)7890



# ソフトなカード。

MEGASOFTのPC-9801シリーズ用拡張ボードシリーズです。  
ソフトハウスならではのノウハウをこめて、好評提供中です。

## STAR-CARD (EM/3STAR)

- 280上位互換の8ビットCPUHD64180 (6MHz)
- 64KBのスタティックRAM (バックアップ機能付)
- 38.4KBPSの拡張RS-232C、タイマ、DMAを用意
- 拡張コネクタを利用してメモリ、I/Oの拡張が可能
- MS-DOS V3.1上にOS統合開発環境を提供するソフトウェアを標準添付

128,000円

## SEC-1 (STAR EXTEND CARD-1)

- STARの拡張2階立て部分にユーザー独自の機能を試作開発する場合に使用する
- STAR-CARD専用ユニバーサルカード
- 機能拡張用コネクタを利用して、HD64180、PC-9801の両方のバスを利用可能

9,800円

## MSC-3 (開発中)

- STARFAX上で鮮明な32ドット明朝体フォントを利用するための拡張機能カード
- イメージスキャナ用パラレルインターフェース装備
- STARFAXやRAMディスクを自動起動するオートスタート機能

近日発売

## MSC-2

- イメージスキャナ (PC-IN501/502/503など) を接続可能 (ソフトウェアが別途必要)
- MS-DOSで5" 2Dディスクドライブ (ソフトウェアが別途必要) を使用可能に (PC-80S31など) を使用可能にするデバイスドライバソフト付きのMX-2 (30,000円) も発売中

10,000円

## SFC-10 (STARFAX)

- PC-9801シリーズ (PC-98XA/XLを含む) でファクシミリとの双方向イメージ通信を実現
- CCITT G3規格対応の4800/2400BPSのモデム
- 自動発着信可能なNCU装備
- モジュラージャックで電話回線、電話機に接続。
- 1本の回線を電話とファクシミリで共用可能

99,800円

## MSC-1 (OEM供給)

- ソフトウェアの不正コピー使用を防止するプロテクトボード (PC-98XL/XAを含む全PC-9801シリーズに対応)
- 独自の専用PALチップ方式により1枚のボードで最大12種類のソフトウェアハウス、システム
- このボードはソフトウェアハウス、システムハウス向け専用の製品で一般の方への販売は行いません。

使いやすさと信頼性

# MEGASOFT

メガソフト株式会社

〒564 吹田市江の木町16-9 近藤ビル6F  
TEL 06-386-2058(代) FAX 06-386-2123



# トランジスタ技術 SPECIAL

No.3

CONTENTS

## FEATURES

16ビット・パソコンを使いこなすためのハード&amp;ソフト

## PC9801と拡張インターフェースのすべて

第一章	PC9801シリーズの内部構成 ● 秋葉澄伸	2
	①PC9801の主要ハードウェア	3
	②PC9801シリーズのメモリ構成と割り込み	14
	③PC9801シリーズの周辺ハードウェアとBIOS	23
	④PC9801VXについて	31
第二章	PC9801シリーズの拡張スロットの詳細 ● 岸本英一	36
	①拡張基板の外形と信号線の意味	36
	②拡張スロットの電氣的仕様	41
	③拡張ボードの設計例1 汎用ICによるI/Oポート	44
	④拡張ボードの設計例2 80系周辺LSIの接続	51
第三章	PC9801用1Mバイト・メモリ・ボードの作り方 ● 秋葉澄伸	53
	①PC9801用1MバイトDRAMボードの設計	53
	②ハードウェアの設計	56
	③1MバイトDRAMボード用デバイス・ドライバの作成	60
第四章	PC9801用A-D/D-A変換ボードの作り方 ● 坂本雄児/塚原英一	67
	①PC9801用A-D変換ボードの製作	67
	②PC9801用D-A変換ボードの製作	82
第五章	PC9801用メカトロニクス・インターフェース・ボードの作り方 ● 形柳正弘/塚原英一	87
	①アイソレート入出力ボードの製作	87
	②アップ/ダウン・カウンタ・ボードの製作	90
	③4ch直流SSR出力ボードの製作	95
	④ステッピング・モータ制御ボードの製作	98
第六章	PC9801用グレードアップ・インターフェース・ボードの作り方 ● 石田優作/大久信広/沢井克敏	106
	①マウス・インターフェース・ボードの製作	106
	②音声処理ボードの製作	112
	③ロジック・アナライザの製作	120
第七章	PC9801のグラフィックの詳細と活用法 ● 阿部英志	139
	①PC9801のソフトウェア構成とBIOS	139
	②グラフィックに関するLIOのサポート	141
第八章	MS-DOS上でのアセンブラ・プログラミング ● 山本強/松平正年	152
	①PC9801のオペレーティング・システム	152
	②MASMによるプログラミング	154
	③MASMの実行方法	158
	④MS-LINKとMS-LIBの使い方	164
	⑤MS-FORTRANで使えるユーティリティ	168







## FEATURES

PC9801は、8086CPUを用いた標準的なパーソナル・コンピュータです。現在は、V30という上位CPUが使われていますが、基本的な構成になっているためあらゆる用途に使用することが可能です。本書では、このPC9801の拡張インターフェースに焦点を当て、様々な応用法について具体的な例で解説しました。

# 16ビット・パソコンを使いこなすためのハード&ソフト PC9801と拡張インターフェースのすべて





# PC9801シリーズの内部構成

本章では、PC9801シリーズの内部構成と基本入出力プログラム(BIOS)について解説します。パソコンをブラック・ボックスとして使用するのが一般的ですが、少し複雑な要求をすると内部の知識が必要になってきます。

現在では、パーソナル・コンピュータはあらゆる仕事のサポートに利用できます。ワープロとして、データ・ベースとして、その他、データ通信の端末として、その活用法は無限にありそうです。これらの用途はすべて市販のソフトを利用できるものです。

しかし、例えば、実験室でデータを集めたいとか、機械をコントロールするためにパソコンを利用したいというときはどうすればよいのでしょうか。

そのようなソフトは自分で作らなければなりませんし、ハードウェアのインターフェースも必要です。

一昔前までは、ソフトも含めてパソコンに近いものを自作することが最善と考えられていましたが、現在ではそういった手段はあまり有効とはいえなくなってきました。

本書で取り上げるPC9801シリーズには、そのために拡張インターフェースのスロットがあります。本書

では、これを有効に活用するために必要な知識をまとめました。

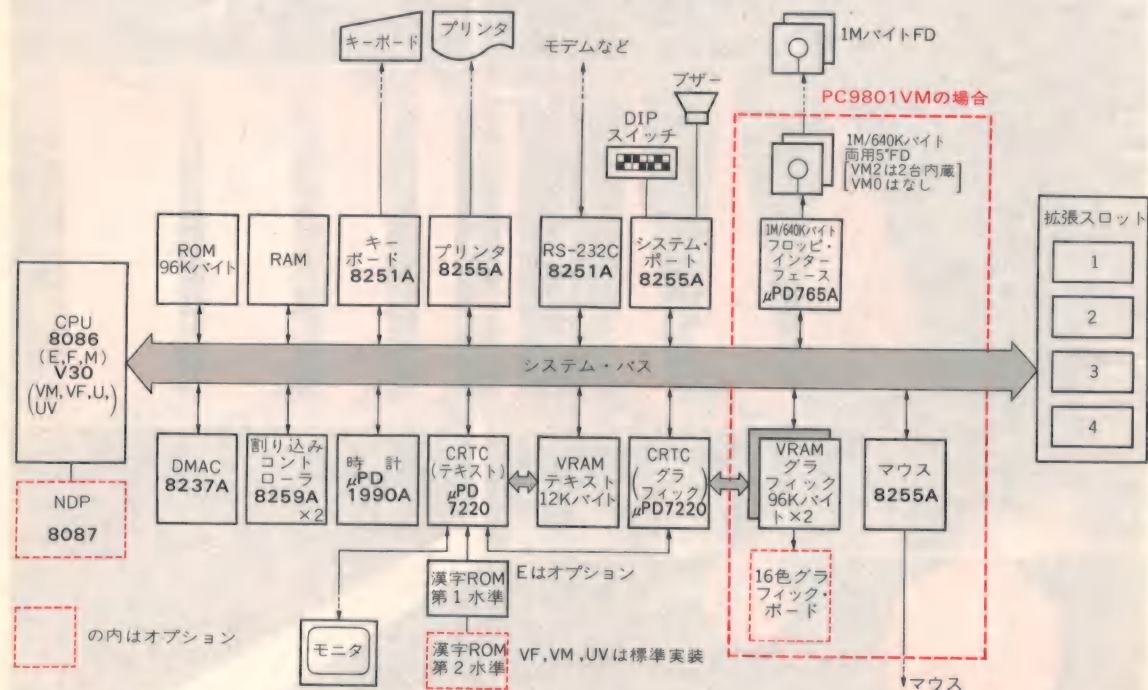
また、本章での解説を理解することにより、16ビット・パーソナル・コンピュータがどのように構成されているのかを知っていただければ幸いです。

## ●PC9801シリーズの概要

PC9801シリーズには、初代のPC9801、E、F、M、VF、VM、U2、UV2などに加え、最新のVX、XL、LTがあり、さらにファクトリ用のFC9801と実に多くの機種があります。その他にも、例えばVMではフロッピー・ディスク装置を内蔵していないVM0、2台を内蔵したVM2、フロッピー・ディスク2台とハード・ディスクを内蔵したVM4などのバリエーションがあります。

しかし、ハードウェア面からは、

〈図1-1〉 PC9801シリーズの内部ブロック図





- ① 初代PC9801
- ② PC9801E/F/M
- ③ PC9801VF/VM/U 2 /UV 2 /VX
- ④ PC9801XA/XL
- ⑤ PC9801LT
- ⑥ FC9801

と大きく6種類のグループに大別することができます。

PC9801シリーズは、初めて登場したときは8086を使った16ビット・システムでしたが、Vシリーズからは8086の上位CPUであるV30が使用され、最新のVXシリーズ、XLシリーズではV30に加えて80286も使用されるようになりました。

しかし、細かな仕様および性能は向上しましたが、基本的な内部構成はすべての機種も同じと考えることができます。そこで、まず初めに、**PC9801を8086の基本システムと考えて、内部構成を調べていく**ことにします。

ここでは、PC9801F 2 を代表として取り上げて、そのハードウェアについて解説します。

ただし、これらのPC9801シリーズでは、初代のPC9801を除くと内部にたくさんのカスタムLSIが使われているため、信号線の内容を完全に調べることは困難になってきています。特に最新のVX/XL、VM21などでは、フラット・パッケージのカスタムLSIやPALなどが多く使用されるようになっていきます。

こうなると、ユーザが本体内部に手を加えることはほとんど考えられず、必要なのは拡張コネクタから外

側の仕様だけになります。

## 1 PC9801の主要ハードウェア

PC9801の基本ブロック図を図1-1に示します。PC9801はCPU周辺についてみれば、8086/V30をCPUとして設計された標準的なコンピュータです。したがって、内部バス構成も使用チップも8086の標準的なインターフェースとなっています。

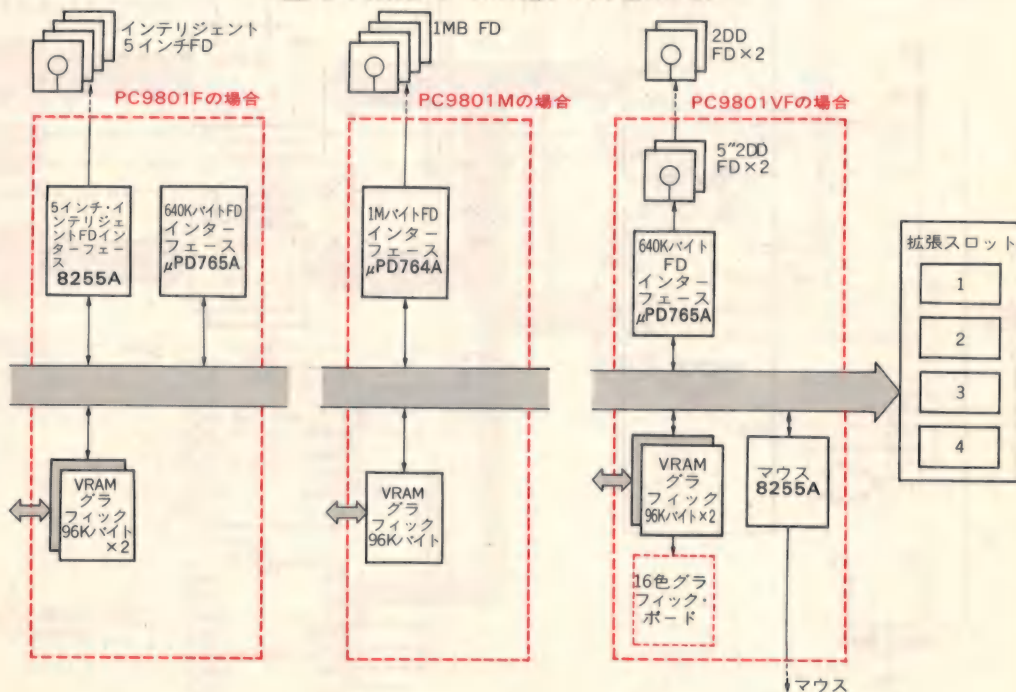
### 1-1 CPU周辺のハードウェア

8086CPUは、バス・コントローラに8288を使用して、マキシマム・モードで動作します。8288から出力されるメモリやI/Oへのバス信号のうち、直接使用されているのはMRDCとAMWC<sub>0</sub>だけで、それぞれ拡張バスのMRC<sub>0</sub>とMWC<sub>0</sub>に対応しています。他の信号はほとんどがカスタムLSIによって作られています。

CPUの周辺回路は図1-2のようになっています。アドレスのラッチはLS373が3個使用されていて、8288のALEにより20本のアドレスとBHE<sub>0</sub>がラッチされます。DMA動作時やCPUKILL<sub>0</sub>信号などにより、CPUがバスを開放する場合には、アドレス・ラッチのLS373のOCによってバスがCPUから切り離されます。OCをコントロールしている信号もカスタムLSIが作っています。

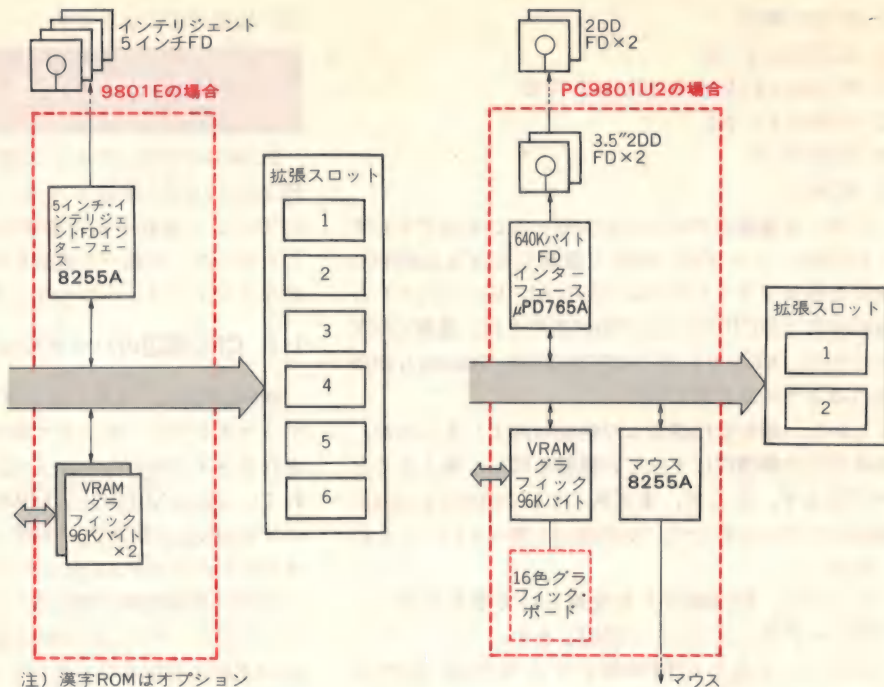
割り込みコントローラ8259Aは他のI/Oデバイスとは異なり、アドレス・ラッチやデータ・トランシーバ

〈図1-1〉 PC9801シリーズの内部ブロック図(つづき)

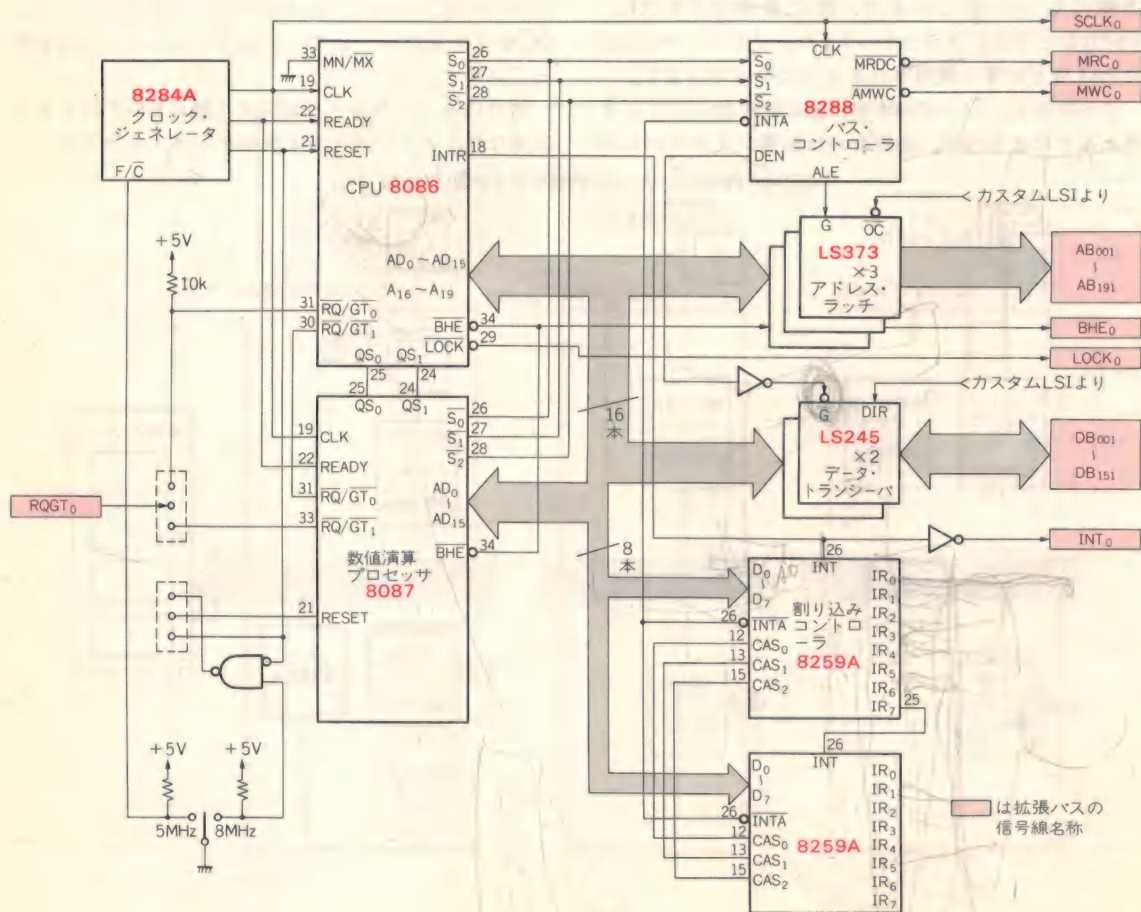




〈図1-1〉  
PC9801シリーズの  
内部ブロック図  
(つづき)

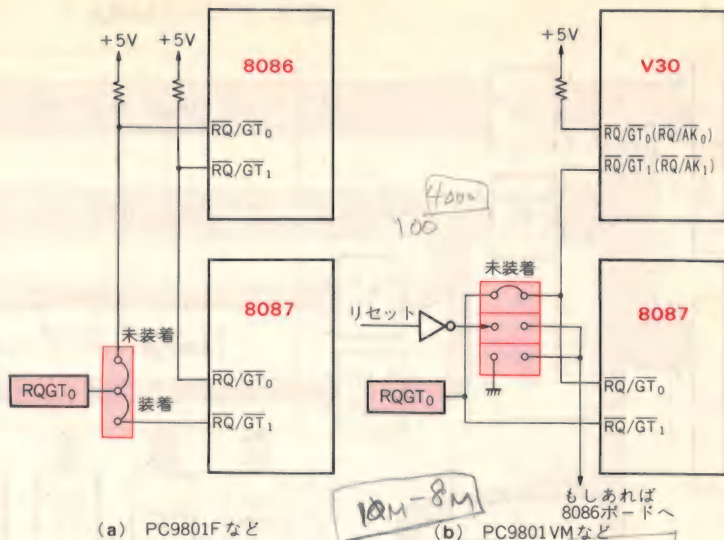


〈図1-2〉 CPU周辺の回路構成





〈図1-3〉 リクエスト・グラント信号



よりもCPU側に接続されています。割り込み受け付け時の8259AとCPUとの間のバスの動作は、通常のリード/ライトとは大きく異なりますので、データ・トランシーバの制御が複雑にならないようにするために、CPUと直接に接続されています。

CPUと直接に接続されるものには、もう一つ数値演算プロセッサ8087があります。8087はオプションですが、装着時にはリクエスト/グラント信号の接続を変える必要があります。

拡張バスのRQGT<sub>0</sub>信号は、8087が未装着時にはCPUの $\overline{\text{RQ}}/\text{GT}_0$ に直接入り、8087装着時には8087の $\overline{\text{RQ}}/\text{GT}_1$ につながれます。図1-3にジャンパの接続を示します。VMでは、8087未装着時にはRQGT<sub>0</sub>信号はCPUの $\overline{\text{RQ}}/\text{GT}_1$ に入ります。

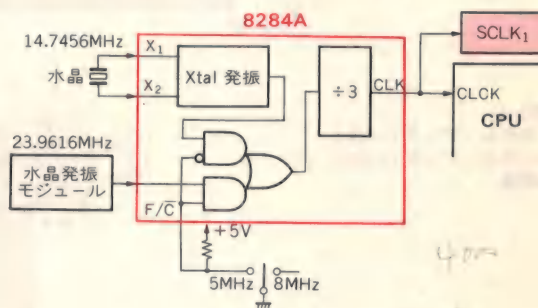
8 MHzと5 MHzのクロックの切り替えは、図1-4に示すようにクロック・ジェネレータ8284Aに入る23.9616MHzの信号と、8284A自身が発振している14.7456MHzを、クロックのスイッチからの信号で切り替えています。8284Aのクロック出力は、拡張バスのSCLK<sub>1</sub>信号に直接出しています。

VM の場合には、8284A が使用されておらず、 $\mu$ PD71011 がクロックを作っています。 $\mu$ PD71011 は内部で 3 分周ではなく 2 分周を行います。したがって、もとの発振周波数は 10MHz 時には 19.6608MHz、8MHz 時には 15.9744MHz となります。

## 1-2 ローカル・データ・バスについて

データ・バスの構成を図1-5に示します。I/O空間のチップはすべて8ビットなので、16ビットのデータ・バスを上位と下位の8ビットにあらかじめ分けてあります。そこで、分けられたデータ・バスを上位と下位のローカル・データ・バスと呼んで、今後はシステム・

〈図1-4〉 クロック・ジェネレータ



データ・バスと区別します。

拡張コネクタのDB<sub>001</sub>からDB<sub>071</sub>に対応するローカル・データ・バスをLDB<sub>001</sub>からLDB<sub>071</sub>(下位のローカル・データ・バス), DB<sub>081</sub>からDB<sub>151</sub>に対応するローカル・データ・バスをLDB<sub>081</sub>からLDB<sub>151</sub>と、頭にLをつけて区別します。ローカル・データ・バスには、図1-6のようにLS245によるバス・ドライバが入っています。

また、拡張バスにはローカル・データ・バスは出ていませんので、拡張ボード上のデバイスもすべてシステム・データ・バスに乗ることになります。

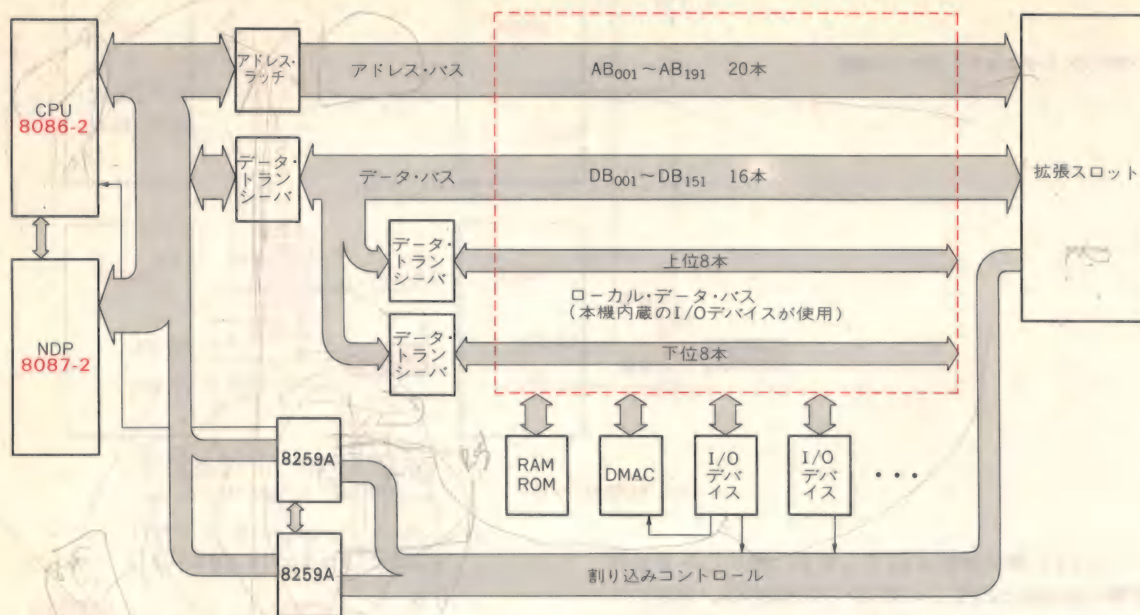
それに対し、本体に装備されているI/Oデバイスは、ほとんどがローカル・データ・バスに接続されています。

ただし、フロッピー・ディスク・コントローラ $\mu$ PD765Aなどは8ビットのI/Oデバイスですが、ローカル・データ・バスではなくシステム・データ・バスに接続されています。

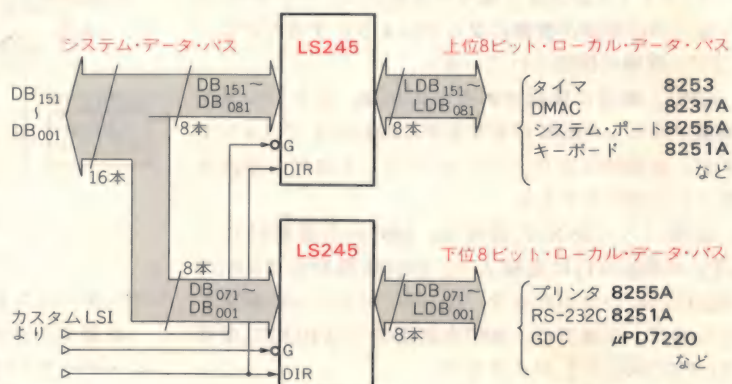
その理由は、DMAなどを使用する場合は、ローカル・データ・バス用のバッファのコントロールが複雑になるために、ローカル・データ・バスを使用してい



〈図1-5〉 データ・バスの構成



〈図1-6〉  
システム・データ・バスと  
ローカル・データ・バスと  
の関係



ないようです。

### 1-3 DMAコントローラの周辺回路

図1-7にDMAコントローラの周辺回路を示します。DMAC8237Aは、I/O空間にあるデバイスとして、CPUからコマンドを受けたりステータスを返したりして、レジスタの値を読み書きされます。

そのために、データD<sub>0</sub>からD<sub>7</sub>までがローカル・データ・バスの上位LDB<sub>081</sub>からLDB<sub>151</sub>までに、アドレスA<sub>0</sub>からA<sub>7</sub>までが双方向バス・ドライバLS245を通してアドレス・バスのAB<sub>011</sub>からAB<sub>081</sub>に接続されています。このときCPUからDMACのレジスタ群は、I/O空間の奇数番地に見えることになります(図1-8)。

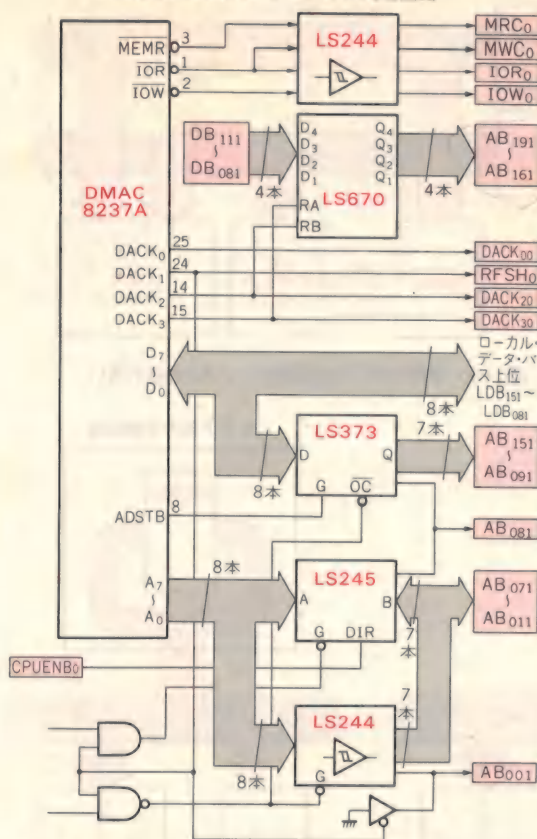
DMACがDRAMのリフレッシュの動作をするときも、図1-8と同様にLS245を通してAB<sub>011</sub>からAB<sub>081</sub>の8本にアドレスが出力されます。このときAB<sub>001</sub>は“1”に固定されています。

DMACが、I/OデバイスからのDMAのリクエストに対してアドレスを発生する場合は、A<sub>0</sub>からA<sub>7</sub>がLS244を通してAB<sub>001</sub>からAB<sub>071</sub>に接続されます(図1-9)。前記のLS245を通したものと、アドレスが1本ずれていることに注意してください。DMACは、1バイトずつの転送を行うためにAB<sub>001</sub>からAB<sub>071</sub>の8本を直接出力し、AB<sub>081</sub>からAB<sub>151</sub>の8本はLS373にラッチして出力できます。

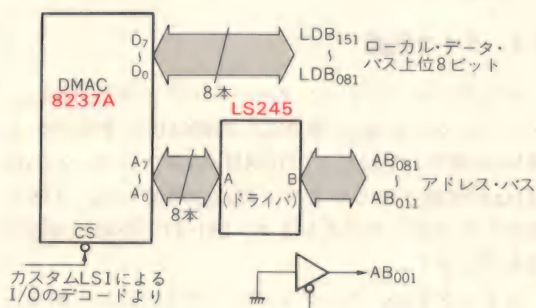
この16本で64Kバイトのアクセスができますが、AB<sub>161</sub>からAB<sub>191</sub>の4本は、LS670がラッチしていたものが使用されます。ですからDMACだけでは、1Mバイトを64Kバイトの16個のブロックに分けると、**ブロックの境界をまたぐような転送はできません**。64Kバイトの境界をまたぐ場合は2度に分けて、バンク・レジスタに最上位4ビットをラッチし直す必要があります。

DMAのバンク・レジスタにはLS670が用いられてい





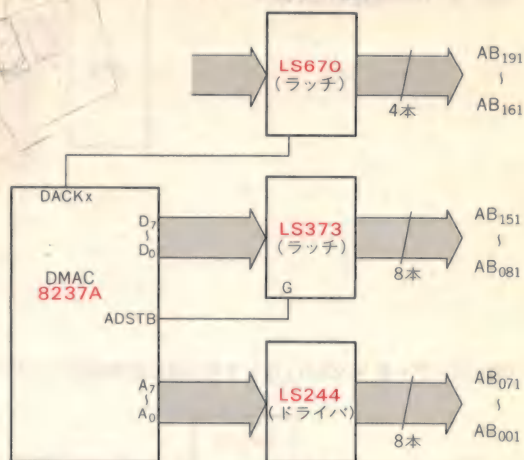
〈図1-8〉 CPUからのアクセスとリフレッシュの場合



ますが、バンク・レジスタのI/Oマップを見ると図1-10のようになっており、チャネルの順番がばらばらでとても不自然です。

これは図1-11に示すように、バンク・レジスタへの書き込みの際は、AB<sub>021</sub>とAB<sub>011</sub>がLS670のセレクトに入っていますから4ビットのアドレスを4個ラッチできます。その4ビットがAB<sub>161</sub>からAB<sub>191</sub>に出力されるときは、DMACのアクノレッジが、リードのセレクトに図1-11のように入りますから、図1-12のようになって、図1-13のようなバンク・レジスタの配置に

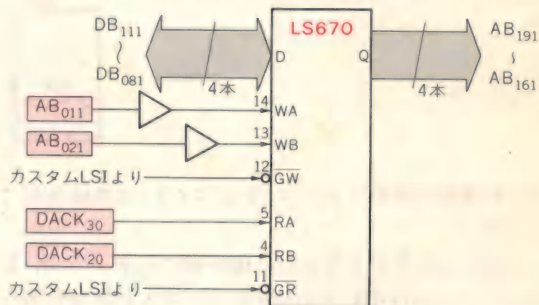
＜図1-9＞ DMA転送中の場合



〈図1-10〉 DMAのバンク・レジスタのI/Oアドレス

I/Oアドレス	内 容
21h	使用できない
23h	チャネル 2 のバンク bit 3 ~ bit 0
25h	チャネル 3 のバンク bit 3 ~ bit 0
27h	チャネル 0 のバンク bit 3 ~ bit 0

〈図1-11〉 LS670に入る信号線



〈図1-12〉  
LS670のチャネル選択

	(RB) DACK <sub>20</sub>	(RA) DACK <sub>30</sub>
チャネル0動作時	1	1
チャネル2動作時	0	1
チャネル3動作時	1	0

〈図1-13〉  
バンク・レジスタの配置

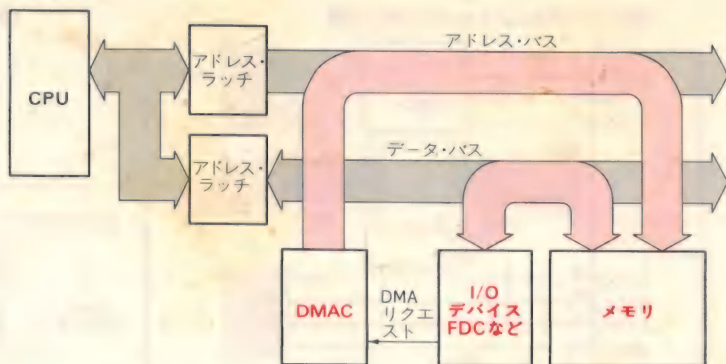
RA	RB	内 容
0	0	—
0	1	チャンネル 3 動作時
1	0	チャンネル 2 動作時
1	1	チャンネル 0 動作時

なります。チャンネル1のアクノレッジは、そのままRFSH<sub>0</sub>信号として拡張バスにも出ています。

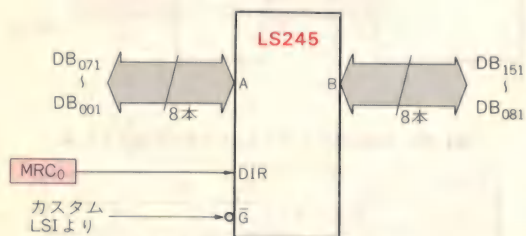
DMAを使用するI/Oデバイスは、フロッピー・ディスクやハード・ディスクのコントローラですが、それら



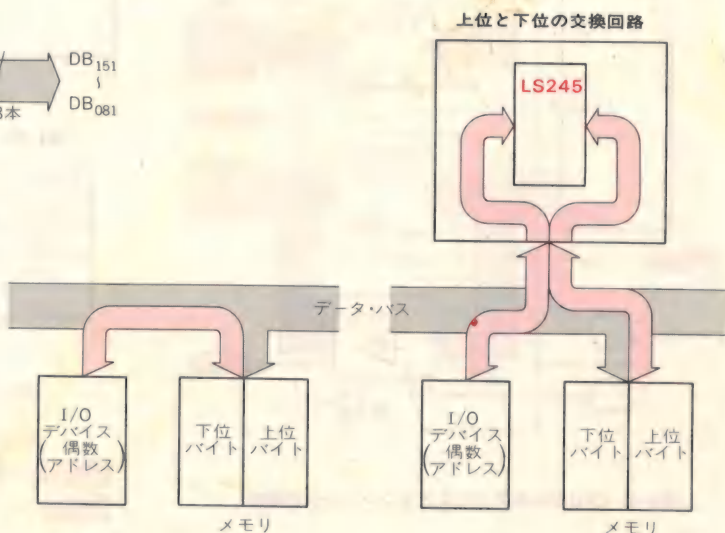
〈図1-14〉 DMA転送のしくみ



〈図1-15〉 データ・バスの上位と下位を結ぶ双方向性バッファ



〈図1-16〉 奇数アドレスと偶数アドレスの転送の違い



はI/O空間の偶数アドレス(下位バイト)に接続されています。

DMAを使用する場合は、図1-14に示すように、I/OデバイスがDMACにDMAのリクエスト信号を送ります。すると、DMACはアドレスを発生させ、I/Oデバイスとメモリの間でデータ・バスを通じて直接にデータの転送が行われます。

ここで注意しなければならないのは、**フロッピー・ディスク・コントローラ**などは、データ・バスの下位8ビットに接続されており、そのままでは**奇数アドレス(上位バイト)のメモリ**とは、データの転送が物理的にできないということです。そのため、図1-15のような回路がデータ・バスに接続されており、下位DB<sub>001</sub>からDB<sub>071</sub>と上位DB<sub>081</sub>からDB<sub>151</sub>とを結んで、奇数アドレスとのデータ転送を可能にしています。奇数アドレスと偶数アドレスの違いを図1-16に示します。

このような理由から、**DMAを使用するデバイス**は**すべて偶数アドレスに存在**しています。メモリとメモリの間のDMA転送は、DMACのレジスタが奇数アドレスにあるため使用できません。

## 1-4 タイマ回路

プログラマブル・インターバル・タイマ8253には、クロックが8 MHzの場合に1.9968MHz、Fなどの5 MHzの場合とVMなどの10MHzの場合には、2.4576 MHzが供給されています。8253の内部には、3個の独立したカウンタがあります。図1-17に8253の周辺回路を示します。

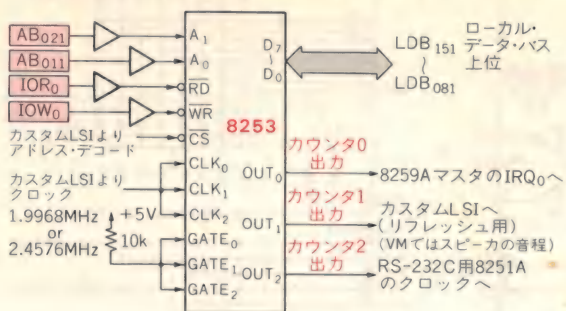
カウンタ0は、インターバル・タイマとして割り込み発生用にも用いられます。

カウンタ1は機種により異なり、Fなどではメモリ・リフレッシュ用に用いられていますが、VMなどからはスピーカの発振音源として使用されるようになっています。Fなどではリフレッシュの周期は28.5μsにセットされています。

したがって、DRAMの256ロウ・アドレスのリフレッシュに7.3msかかりますから、拡張メモリ・ボードなどの設計時にはリフレッシュ・サイクルは4msでは不十分です。それに対し、VMなどでは4msの設計で大丈夫です。VMではリフレッシュは、カスタムIC



〈図1-17〉 タイマLSI 8253の周辺回路

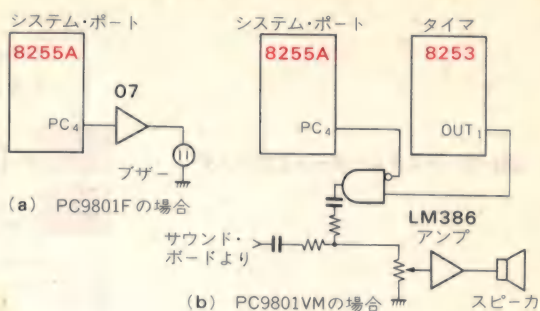


〈図1-18〉 8253の機種によるI/O アドレスの違い

I/Oアドレス		内 容
71h	R/W	カウンタ0 インターバル・タイマ
73h	R/W	カウンタ1 リフレッシュ・サイクル
75h	R/W	カウンタ2 RS-232C用クロック
77h	W	モードのセット

(a) PC9801Fの場合

〈図1-19〉 ブザー回路



I/Oアドレス		内 容
71h	R/W	カウンタ0 インターバル・タイマ
73h	R	カウンタ1 スピーク音程
3FDBh	R/W	カウンタ2 RS-232C用クロック
77h	W	モードのセット
3FDFh	W	モードのセット

(b) PC9801VMの場合

からの約17μsの周期の信号により行われているからです。

カウンタ2は、RS-232C用のクロックを発生しています。また、カウンタ1の機種による互換性をとるために、I/Oのアドレスに違いがあります。すなわち、同じ8253のカウンタ1を使用しているにもかかわらず、図1-18に示すようにI/Oのデコードを変えてあります。

図1-18からわかるように、Fなどでは73h番地に値を書くことでリフレッシュ・サイクルを変えることができますが、VMの73h番地に値を書き込んでもスピーカの音程は変化しません。

音程を変化させる場合は、3FDBh番地に書く必要はありません。図1-18の71h、73h、75h、77h番地は、アドレスの下位8ビットしかデコードしていないという意味で、3FDBh、3FDFhは

アドレスの16ビットをデコードしているという意味です。

図1-19にブザーの回路を示します。

## 1-5 システム・ポート

システム・ポートには8255Aが1個使用されていて、各種のI/Oデバイスの情報を扱っています。

ポートAは入力で、DIPスイッチSW<sub>2</sub>の情報を読み込んでおり、システムが立ち上がる時の状態を決定します。

ポートBも入力で、RS-232C用の信号やカレンダーLSI μPD1990Aのデータを読み込むために使用します。

ポートCは出力で、プリンタ・ストローブのマスク、ブザーのコントロール、RS-232Cの割り込み許可な

# 別冊 トランジスタ技術

B5判 184頁

SPECIAL No.2

特集 作りながら学ぶMC68000 16ビットMPUとその周辺LSIを使いこなすためのハード&ソフト

好評発売中

定価 1,500円

二色刷



コンピュータらしいといわれる16ビット・マイクロプロセッサMC68000を理解するには、1から作ることです。時間がない人は、本書を読んで学んでください。

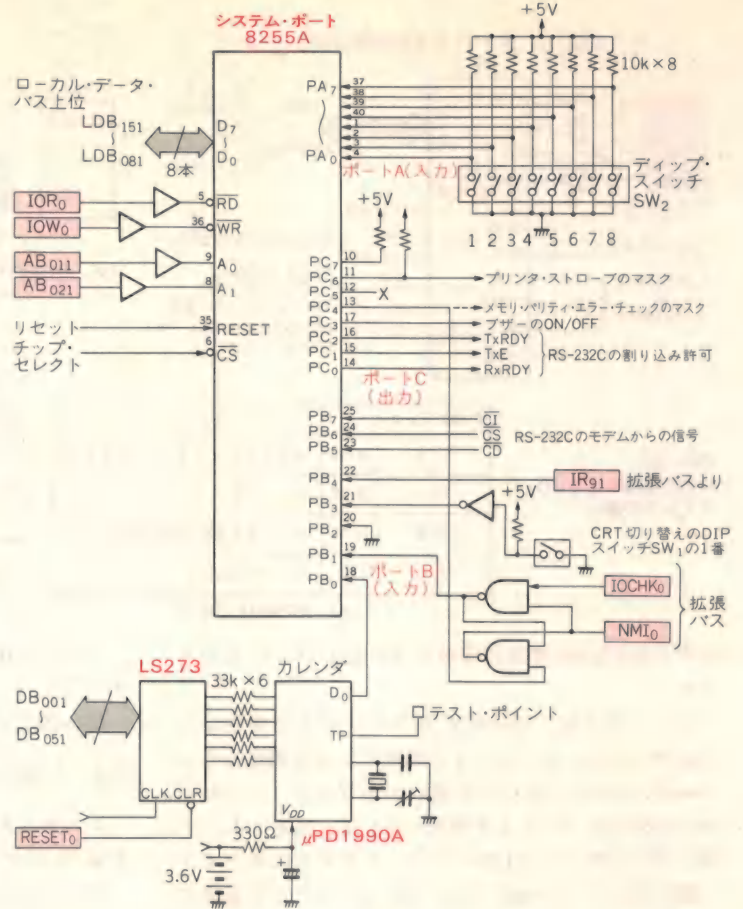
## 目次

MC68000とCPUボードの製作／モニタ・プログラムの搭載／ターミナルとの接続／命令セットとアセンブラ文法／I/Oボードの製作／フロッピー・ディスク・コントロール・ボードの製作／フロッピー・ディスクとのインターフェース／FORTHの移植手順／FORTHの使い方とツール／RAMボードの製作／CP/M-68Kの移植(1)／CP/M-68Kの移植(2)／CP/M-68Kの移植(3)／C言語とシステムのグレードアップ

CQ出版社



〈図1-20〉 システム・ポートとカレンダー



(注) ONは“0”, OFFは“1”が見える。

〈図1-21〉 システム・ポートの各ポートの内容

(a) ポート A (入力) ディップ・スイッチ SW<sub>2</sub>

ビット 7	}	未使用
6		
5		
4	ON: メモリ・スイッチを保存	OFF: メモリ・スイッチを初期化
3	ON: 1 画面 25 行	OFF: 20 行で GDC を初期化
2	ON: 1 行 80 文字	OFF: 40 文字で GDC を初期化
1	ON: ターミナル・モード	
0	ON: 初期イニシャライズの動作の一部をパス (通常は OFF)	

(b) ポート B (入力)

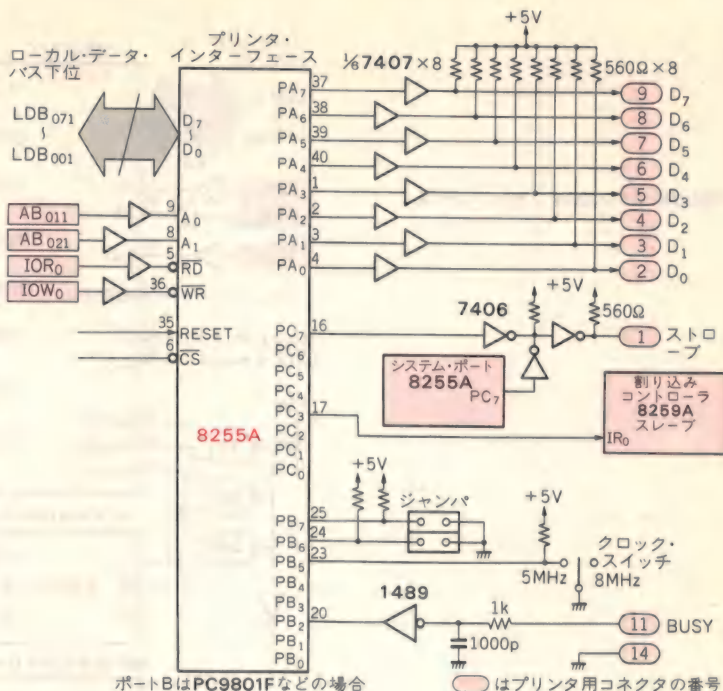
ビット 7	CI 信号
6	RS-232C の CS 信号
5	
4	拡張バスの IR <sub>91</sub> (INT <sub>3</sub> : 5 インチ HD)
3	CRT のタイプ, 1 = 高解像度 (400 ライン) 0 = 低解像度
2	未使用
1	拡張 RAM のパリティ・エラー
0	カレンダー μPD1990A のデータ読み出し

(c) ポート C (出力)

ビット 7	未使用
6	プリンタのストロブ信号のマスク
5	未使用
4	メモリのパリティ・チェック Enable (VM では未使用)
3	ブザー制御
2	TxRDY
1	
0	
	RS-232C の TxE による割り込み許可
	RxDY



〈図1-22〉 プリンタ・インターフェース



ポートBはPC9801Fなどの場合

○はプリンタ用コネクタの番号

〈図1-23〉 プリンタ・インターフェース  
8255AのポートBの内容(42h)

(a) PC9801Fなどの場合

ビット	7	6	5	4	3	2	1	0
0	PC9801	0	E/F/M, VM/VF	1	U2	機種	の	フラグ
5	0	5 MHz	1	8 MHz	クロック	周波数		
4					未使用			
3					未使用			
2					プリンタ	の	BUSY	信号
1					未使用			
0					未使用			

(b) PC9801VMなどの場合

ビット	7	6	5	4	3	2	1	0
0	PC9801	0	E/F/M, VF/VM	1	U2	機種	フラグ	
5	0	10MHz	1	8 MHz	クロック	周波数	を選択	
4	0	LCD	プラズマ・ディスプレイ	を使用				
3	0	16色	表示機能, 高速	描画機能	を使用			
2		プリンタ	の	BUSY	信号			
1	0	8086-2	1	PD70116 (V30)	CPU	の	タイプ	を選択
0	0	VM	1	VF	機種	を選択		

(c) PC9801VXの場合

ビット	7	6	5	4	3	2	1	0
0	PC9801	0	VX	機種	の	フラグ		
5	0	10MHz (V30)	1	8 MHz (V30, 80286)	クロック	周波数	を選択	
4					VM	と同じ		
3					VM	と同じ		
2					VM	と同じ		
1	0	80286	1	V30	CPU	の	タイプ	を選択
0								

どに使用されています。このシステム・ポート部、回路構成を図1-20に示します。また、システム・ポートの内容を図1-21に示します。

ポートAとディップ・スイッチSW<sub>2</sub>との関係で注意しなくてはならないのは、SW<sub>2</sub>がONになると対応するポートAのビットが“0”となり、SW<sub>2</sub>がOFFになると“1”になるということです。

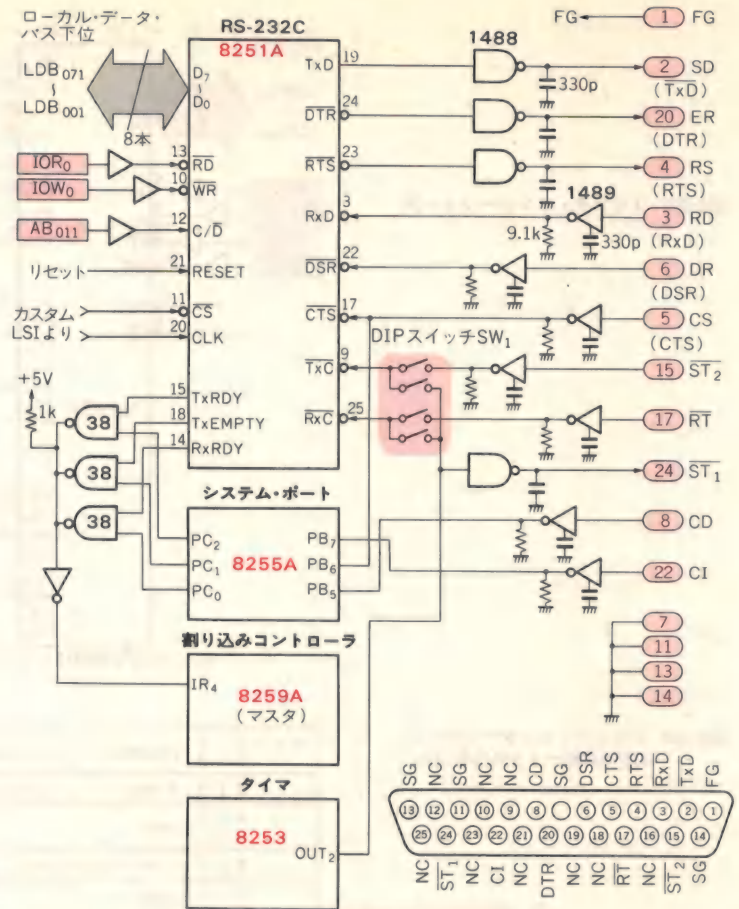
## 1-6 プリンタとのインターフェース回路

プリンタ・インターフェースには汎用の8255Aが使用されています。図1-22のようにシステム・ポートからプリンタのストローブ信号をマスクしたり、割り込みコントローラへ割り込みをかけたりできるようになっています。

また、ポートBは入力で、プリンタのビジー信号の



〈図1-24〉 RS-232Cインターフェース



ほかにシステムのタイプなどの情報も得られます。ただし、この部分については機種により異なるので注意してください。図1-23に8255AのポートBの内容を示します。

### 1-7 RS-232Cのインターフェース回路

RS-232Cのインターフェースは、図1-24に示したように8251Aを使用した標準的な回路です。

回路を見ればわかるように、システム・ポートのポートBでモデムのコントロール信号が読めるようになっており、ポートCから割り込みの許可の制御も可能になっています。

内部と外部のクロックの切り替えは、図1-25のように機種により異なります。

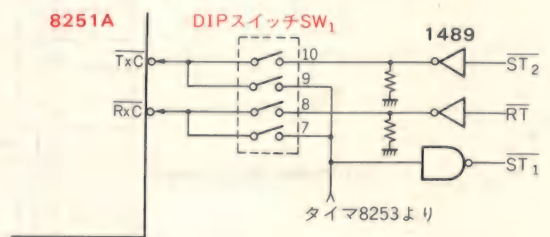
PC9801Fなどでは4個のディップ・スイッチが使用されていましたが、VMからは1個のディップ・スイッチで簡単に切り替えが可能になりました。

### 1-8 キーボード・インターフェース回路

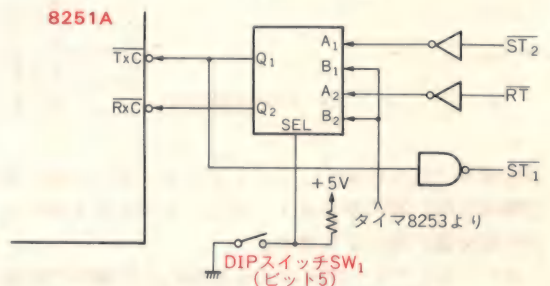
キーボードからは、信号線4本と電源、アースの計6本が、コール・コードにより出ています。

キーボードの100個のキー(VMなどは101個)が押さ

〈図1-25〉 RS-232Cクロック切り替え部分



(a) PC9801Fなどの場合



(b) PC9801VMなどの場合

れたとき(MAKE)と離れたとき(BREAK)にそれぞれ信号が8251Aに送られます。8251Aは信号を受け



**【図1-26】 キーボード・インターフェース**  
 この図は、8251Aのピン構成と外部回路を示しています。8251Aのピンは、左側にバス上位（D<sub>7</sub>～D<sub>0</sub>）とバス下位（LDB<sub>151</sub>～LDB<sub>081</sub>）の8本のバス線、そして右側にTx<sub>D</sub>、RxRDY、RTS、Rx<sub>D</sub>、DTR、DSRなどのシリアル通信ピンが配置されています。外部回路には、+5V電源、620Ω、910Ω、1000pFなどの抵抗と容量、そして8259A（割り込みコントローラ）が接続されています。また、キーボード・コネクタの番号（1～8）も示されています。

本体側

キーボード側

+5V

DATA

RDY

RTY

RTS

GND

8251A

8048

LS159

キー・マトリクス

ると割り込みを発生させます。キーボード・インターフェース部の回路構成を図1-26に示します。また、キーボードと本体の接続図を図1-27に示します。

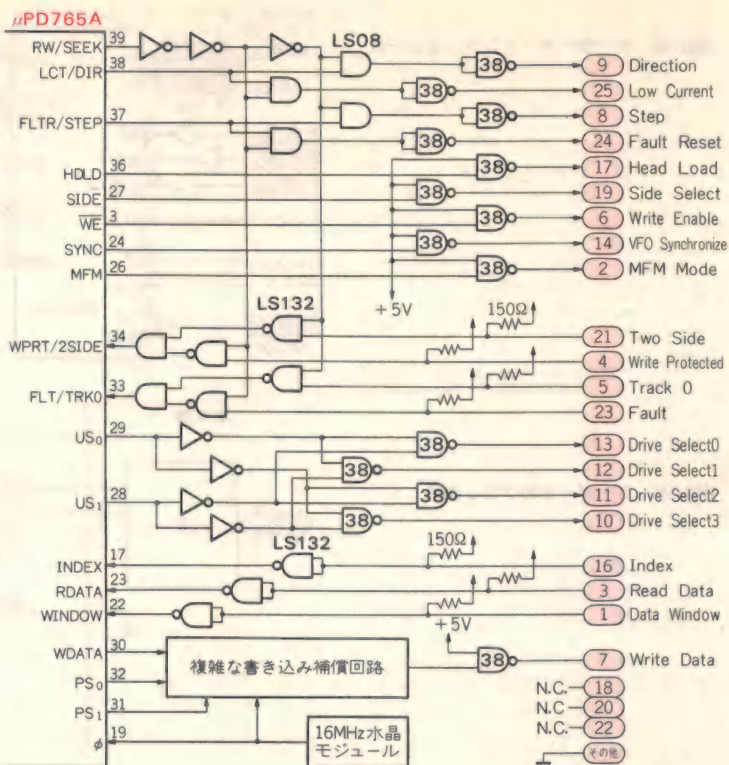
マウスとのインターフェースには8255Aが用いられており、8255Aの先にはマウスの移動量カウンタとマウスのスイッチが接続されています。その他にはタイマICがあり、定期的に割り込みをかけることができます。

また、タイマからの割り込みレベルは、ジャンパによって変更できるようになっています。通常はIR<sub>131</sub> (INT6)になっています。

## 13



〈図1-29〉 1MバイトFDインターフェースと  
μPD765Aの接続



- ① CPUがDMAコントローラをセットする。
- ② CPUがFDCにコマンドを送る。
- ③ FDCがDMAのリクエストをDMAコントローラに送る。
- ④ 1バイトのデータが転送される。
- ⑤ DMAコントローラからFDCにDMAのアクノレッジ信号が送られる。
- ③～⑤を繰り返す。
- ⑥ 転送が終わるとDMATC<sub>0</sub>信号がDMAコントローラからFDCに送られる。
- ⑦ FDCは割り込みをかける。
- ⑧ CPUは通常の処理を続ける。

となります。

拡張バスのDMA関係の信号とFDCとの接続を図1-28に示します。この例は、1Mバイト用フロッピー・ディスク・インターフェースの拡張基板をもとにしたものです。640Kバイト・フロッピー・ディスク・インターフェースには、インターフェース・ボード上にDMAの禁止フラグなどがあり、データ・セパレータなども乗っています。

参考のため、図1-29に1Mバイト・フロッピー・ディスク用のインターフェース回路とFDCとの接続図を示します。自作をしてフロッピー・ディスク・ドライブを接続する場合などに参考になると思います。

## ②PC9801シリーズのメモリ構成と割り込み

ここでは、PC9801シリーズの機種になるべくよらずに、共通に関係しているソフトウェアの一部について解析したものを紹介します。

### 2-1 PC9801シリーズのメモリ構成

PC9801では、8086-2あるいは上位コンパチブルであるV30(μPD70116)という16ビット・マイクロプロセッサを使用していますが、このCPUは1Mバイトまでの物理アドレスを生成することができ、さらにI/O空間をメモリ空間と独立にもっています。

したがって、PC9801シリーズのメモリ構成を簡単に示すと図2-1のようになっています。

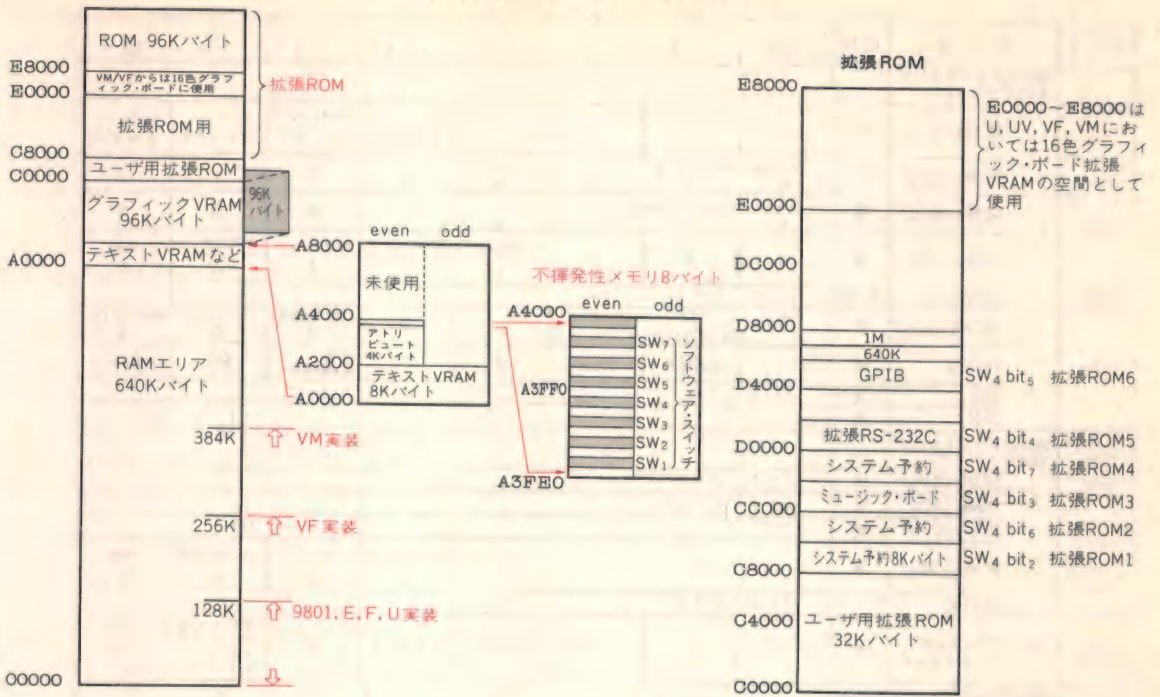
また、1Mバイトのメモリ空間は図2-2に示すように割り付けられています。

PC9801シリーズでは、0～9 F F F F hまでの640KバイトをフリーRAM、その残りをCRT(グラフィックスやテキスト)用のV-RAMやROM-BIOS(N<sub>88</sub> BASICインタプリタをふくむ)などに割り当てています。

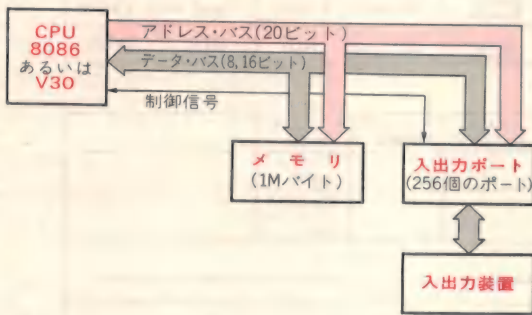
フリーRAMは、E/Fタイプで、128Kバイトが標準実装されていますが、VMでは384Kバイトが標準実装になります。また、グラフィックV-RAMは、同じアドレスをバンク切り替えて使用し、96KバイトのV



〈図2-2〉 PC9801シリーズのメモリ・マップ



〈図2-1〉 PC9801シリーズのメモリ構成



-RAMが2画面分用意されています。そして、E0000hからの32Kバイトは中間色表現のためのV-RAMエリアになっています。

## 2-2 I/O空間の割り付け

CPUのデータ・バスは16本ですが、使用している周辺インターフェースLSIはすべて8ビット用のものが用いられています。したがって、I/O空間では、データ・バスを下位8ビットと上位8ビットの二つに分けて使用しています。

このため、D<sub>0</sub>からD<sub>7</sub>までの下位8本のデータ・バスにつながれたLSIは偶数アドレスとなり、D<sub>8</sub>からD<sub>15</sub>までの上位8本につながれたLSIは奇数アドレスとなります。

したがって、複数のアドレスを占めるチップはアドレスが連続とはならず、奇数または偶数アドレスに一

つ飛びになってしまいます。

そこで、見やすいように偶数アドレスと奇数アドレスに分けたI/Oマップを図2-3に示します。また、これをさらに見やすくしたものを図2-4に示します。

図2-4を見ればわかるように、I/O空間のアドレス・デコードはきわめて不完全です。I/O空間をぜいたくに使ってチップが配列されており、アドレスの間隙にはむだなイメージが見えています。さらにアドレスは、一部の例外を除いて、下位の8ビットしかデコードされておらず、I/O空間の大部分はすでに埋まっています。

例外は、VF/VMから標準装備になったマウスのインターフェースなどで、16ビットのアドレス・デコードがなされています。しかし、それらもユーザが使用可能であつたわずかな空間に入り込んできています。

8ビットしかデコードされていないチップは、上位8ビットを何に指定してもアクセスすることができます。例えば、割り込みコントローラ8259Aは、0000hでも0100hでもFF00hでも、下位8ビットが00hであれば同じレジスタが見えることになります。16ビット・デコードされているチップは、16ビットをすべて指定しなくてはなりません。

このように、ユーザが使用可能なI/O空間は非常に狭いので、自作や市販のボードの間でアドレスの衝突が起りやすく、設計にあたっては注意が必要です。すなわち、容易にI/Oアドレスを変更できるような処置が、ハードウェアとソフトウェアの両面に要求され



〈図2-3〉 PC9801シリーズのI/Oマップ

I/O アドレス	内 容	R/W	bit 7	6	5	4	3	2	1	bit 0
	割り込みコントローラ 8259A(マスタ)									
00h	ICW <sub>1</sub> (11h)	W	0	0	0	1	トリガ・モード 0=EDGE	アドレス・イン ターバル 0=4バイト	0= カスケード	ICW <sub>4</sub> 1=必要
02h	ICW <sub>2</sub> (08h)	W	ベクタ・アドレス 0 0 0			0	1	0	0	0
	ICW <sub>3</sub> (80h)	W	スレープをもつ IRQ 入力 1 0 0			0	0	0	0	0
	ICW <sub>4</sub> (1Dh)	W	0	0	0	SFNM 1	BUFバッファ ア・モード 1	M/Sマスタ 1	AEIOノーマ ルEOI 0	8086 1
02h	OCW <sub>1</sub>	R/W	インターラプト・マスク 0 0 0 1							
00h	OCW <sub>2</sub> (20h)	W	End of Interrupt 0 0 1			0	0	IR LEVEL TO BE ACTED UP ON 0 0 0		
	OCW <sub>3</sub>	W	—	ESMM	SMM	0	1	Poll Command	RR	RIS
	IRR	R								
	ISR	R								
08h 0Ah	割り込みコントローラ 8259A(スレープ)									
08h	ICW <sub>1</sub> (11h)	W	マスタと同じ							
01h 10	ICW <sub>2</sub> (10h)	W	ベクタ・アドレス							
	ICW <sub>3</sub> (07h)	W	スレープ ID							
	ICW <sub>4</sub> (09h)	W					バッファ・モード 1	スレープ 0	ノーマルEOI 0	8086 1
	その他		マスタに準ずる							
20h	カレンダー μPD1990 セット・レジスタ	W			入力データ	クロック	ストロープ	0	0 0 レジスタ・ホールド 0 1 レジスタ・シフト 1 0 1 1 タイム・リード	
30h	RS-232C用8251A データ	R/W	シリアル・データ							
32h	ステータス	R	DSR	SYNDET	FE	OE	PE	TxE	RxRDY	TxRDY
	モード	W	Stop Bits		Even Parity	Parity Enable	Character Length		Baud Rate Factor	
	コマンド		Enter Hunt	Internal Reset	RTS	Error Reset	Send Break	RxE	DTR	TxEN
40h	セントロニクス・プリンタ 8255A Port A	W	データ							
42h	Port B VF, UMの場合 E, Fの場合	R	プリンタ・タイプ			プラズマ・ ディスプレイ	グラフィック 拡張機能	プリンタ BUSY		
			プリンタ・タイプ				プリンタ BUSY			
44h	Port C	W	プリンタ・ ストロープ					IRQ 8251A スレープの IRQ <sub>0</sub>		
46h	モードと コントロール	W								
50h	NMIコントロール・ フリップフロップ	W	NMIを起こさない							
52h		W	NMIを起こす							
60h 6Eh	μPD7220 (テキスト)	R/W								
70h 7Eh	μPD52611 (カスタム)	R/W								
80h 82h	5 インチ・ハード・ ディスク	R/W	データ コントロールとステータス							
90h 96h	1MBフロッピー・ディスク μPD765A									
A0h AEh	μPD7220 (グラフィック)									
C0h C6h	ODA プリンタ 8255A									
C8h CEh	5 インチ2DD μPD765A									
DO DE EC EE FO	ユーザ用といわれ ている部分		21 19 6 8 A C E 0							

(a) 偶数アドレス (D<sub>0</sub>—D<sub>7</sub> につながるI/O)



〈図2-3〉 PC9801シリーズのI/Oマップ(つづき)

I/O アドレス	内 容	R/W	bit 7	6	5	4	3	2	1	bit 0		
	DMAコントローラ 8237A											
01h 03h	Channel 0	R/W	Base Address Register Base Word Count Register									
05h 07h	Channel 1											
09h 0Bh	Channel 2											
0Dh 0Fh	Channel 3											
11h	Status Register	R	CH <sub>3</sub>	Request CH <sub>2</sub>	CH <sub>1</sub>	CH <sub>0</sub>	CH <sub>3</sub>	has Reached TC CH <sub>2</sub>	CH <sub>1</sub>	CH <sub>0</sub>		
	Command Register	W	DACK	DREQ	Write	Priority	Timing	Enable	CH <sub>0</sub>	M-M		
13h	Request Register							Set/Reset Request	Channel No.			
15h	Single Mask Register	W						Set/Clear Mask Bit	Channel No.			
17h	Mode Register		Mode Select	Decrement Increment	Auto initialize	Transfer		Channel No.				
19h	Clear Byte Pointer Flip/Flop											
1Bh	temporary Register	R										
	Master Clear	W										
1Fh	All Mask Register bit	W					CH <sub>3</sub>	CH <sub>2</sub>	CH <sub>1</sub>	CH <sub>0</sub>		
	DMAコントローラ・ バンク・レジスタ											
21h	Channel 2 Channel 3 Channel 0	W	使用できない									
23h									A <sub>19</sub>	A <sub>18</sub>	A <sub>17</sub>	A <sub>16</sub>
25h												
27h												
	システム・ポート 8255A											
31h	ディップSW No. 2 Port A	R				メモリ・ス イッチ初期 化しない	画面 1 : 25行 0 : 20行	一行 1:80文字 0:40文字	1:ターミナル 0:ベーシック	bootの時 INT 1Fh をしない		
33h	Port B	R	$\overline{CI}$	RS-232C CTS	$\overline{CD}$	5インチHD INT3	CRT TYPE	内部RAM バリエイティ	外部RAM バリエイティ	カレンダー 読み出し		
35h	Port C	W	未使用	プリンタ・ ストローブ のマスク	未使用	メモリ・チ ェック・イ ネーブル	ブザー停止	RS-232Cの割り込み許可				
37h	コマンド	W	Port Cのビット操作									
	データ	R										
43h	8251A コマンド・ ステータス	R/W										
	5インチインテリジェント フロッピードライブ 8255A											
51h	Port A	R/W										
53h	Port B											
55h	Port C											
57h	コマンド・ モード	W										
	タイマ8253A											
71h	Counter 0	R/W										
73h	Counter 1											
75h	Counter 2											
77h	Mode Word	W	Select Counter	Read/Load			MODE		BCD			
A1h AFh	漢字 ROM											
C1h CFh	GPIB $\mu$ PD7210											
EDh EFh	ユーザ用											
7FD9 7FDB 7FDD 7FDF	マウス用8255A	R	LEFT		RIGHT		移動量4ビット					
		W	HC	SXY	SHL	INT	0	0	0	0		
		W	8255Aのモード・セット									
8F0B	マウス用タイマ	W	タイマ									

(b) 奇数アドレス(D<sub>0</sub>~D<sub>15</sub>につながるI/O)



ます。

また、複雑なハードウェアの構成となると、16ビットのアドレス空間をデコードすることが望ましいのですが、ハードウェアに負担がかかり、ソフトウェアの面からも入出力命令に直接アドレスが指定できなくなるなどの問題が生じてきます。

FM16 $\beta$ などのI/O空間の構成は、バイト、ワード・アクセスの両方に十分な配慮があり、また8ビット・チップも連続したアドレスに配置されています。

ユーザが安全に使用できるI/Oアドレスは、D0h、D2h、D4h、D6h、D8h、EDh、EEh、EFhぐらいです。ただし、絶対安全とはいえません。安全とされていたD1h~DFhの奇数アドレスに

は、マウスのインターフェースやスピーカの音程設定が入ってきましたし、VX登場からはF0hからFhの偶数アドレスがとうとう使用不能となってしまいました。

BASICから使用することがなければ、E0hからEFhも安全だと思います。PC9801シリーズには、RAMディスクをはじめ、各種のI/Oインターフェース・カードを使用してデータの取り込み処理や外部機器の制御を行うことが多いと思います。

このような場合に、I/Oアドレスの使用は上記のアドレスに集中しており、特にインターフェース・カードを作っているメーカによってデコード方法がさまざま、これにソフトウェアがからむとトラブルも多く

〈図2-4〉  
I/Oアドレスの早見表

EVEN		ODD		EVEN		ODD	
00	8259Aマスタ	01	DMAC 8237A	80	ハード・ディスク	81	
02	イメージ	03		82			
		05			イメージ		
		07					
08	8259Aスレーブ	09		90	8インチFDC	91	カセット8251A
0A	イメージ	0B		92	$\mu$ PD765	93	カセット・コントロール
		0D		94		95	イメージ
		0F		96	イメージ	99	GPIBスイッチ
10						9B	
20	カレンダ $\mu$ PD1990	21	DMAバンク	A0	$\mu$ PD7220 スレーブ・グラフィック	A1	漢字キャラクタ・ジェネレータ
	イメージ	23		A2		A3	
		25		A4		A5	
		27		A6		A7	
			イメージ	A8		A9	
				AA		AB	
				AC		AD	
				AE		AF	
				BO		B1	
30	RS-232C 8251A	31	システム・ポート 8255A				
32	イメージ	33					
		35					
		37					
			イメージ				
40	セントロニクス・プリンタ 8255A	41	キーボード 8251A	C0	ODAプリンタ 8255A	C1	GPIB $\mu$ PD7210
42		43		C2		C3	
44				C4		C5	
46				C6		C7	
	イメージ		イメージ	C8		C9	
				CA		CB	
				CC		CD	
				CE		CF	
				D0		D1	
50	NMI	51	5"インテリジェント FDD用 8255A				
52	コントロール	53					
	イメージ	55					
		57					
			イメージ				
60	$\mu$ PD7220 マスタ・テキスト	61		EO			
62							
64							
66							
68							
6A							
6C	$\mu$ PD52611 CRT マスタ・スライス	71	タイマ8253A				
6E		73					
70		75					
72		77					
74			イメージ				
76							
78							
7A							
7C							
7E							

なります。RAMディスクのようにI/OアドレスはEChで、バンク・レジスタの使用法もほぼ統一されるまでには時間もかかりますし、EChを使用していた他の製品は、今後はアドレスを変えざるを得ません。自作、市販を問わず、I/O空間を使用するボードを設計するときは、

- ① 必ず、16ビット・デコードを行う。
  - ② アドレスは容易に変更できるようにしておく。
  - ③ ボードをサポートするソフトウェアも容易にアドレスを変えられるようにする。
  - ④ 他で使用していないようなアドレスをなるべく使用する。
  - ⑤ 連続したI/O空間が必要な場合は下位8ビットは固定のまま上位8ビットでI/Oアドレスをセレクトするようなデコードをする。
- 以上のような注意をお願いします。

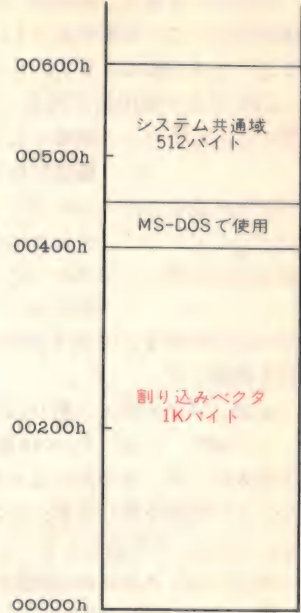
また、どうしても連続して広いI/Oアドレスが必要となる場合は、00000hからC8000hまでのユーザ用拡張ROM領域を使用することも可能です。

### 2-3 PC9801シリーズの割り込み

PC9801シリーズでは、割り込みコントローラ8259Aが重要な働きをしています。そこで、ここでは割り込みについて説明します。

PC9801シリーズのメモリ空間で、00000h番地から003Fh番地までの1Kバイトは、256個の割り込みベクトルとして使用されています。すなわち、0番から255番までの256個がそれぞれ4バイトず

〈図2-5〉  
PC9801のRAMマップ



つの処理プログラムへのベクトルをもっています(図2-5)。

例えば、割り込み3番が発生すると、0000Ch番地から0000Fh番地の4バイトで示されるアドレスのプログラムに制御が移ります。

また、PC9801の割り込みは内部割り込みとハードウェア割り込みに大別できますが、どちらもこの256個のベクトルを使用しています。

#### ▶ 内部割り込み

〈表2-1〉 割り込み番号とその機能

割り込み番号	機 能	割り込み番号	機 能	割り込み番号	機 能
0	除算エラー	14	拡張バスINT <sub>5</sub>	B0 ↓ BF	DISK LIO BASICシステム予約
1	シングル・ステップ割り込み	15	拡張バスINT <sub>6</sub>	C0	ハード・コピー処理
2	NMI割り込み	16	8087	C1	キーボード/CRT
3	1バイト型内部割り込み	17	システム予約	C2	キーボード/CRT
4	オーバフロー	18	キーボード、CRT	C3 ↓ CD	BASICシステム
5	ハード・コピー (COPYキー)	19	RS-232C	CE	ハード・コピー(グラフLIO)
6	STOPキー	1A	カセット、プリンタ	CF	機械語モニタ
7	インターバル・タイマ	1B	DISK	D0	機械語モニタ
8	タイマ	1C	カレンダー、インターバル・タイマ	D1	GPIB
9	キーボード	1D	システム予約	D2	MUSIC
A	CRT	1E	N <sub>88</sub> BASIC	D3	BRANCH4670
B	拡張バスINT <sub>0</sub>	1F	システム予約	D4	RS-232C 第2回線
C	RS-232C	20 ↓ 3F	システム予約	D5	RS-232C 第3回線
D	拡張バスINT <sub>1</sub> (CMT)	40 ↓ 7F	ユーザ開放	D6 ↓ F0	BASICシステム
E	拡張バスINT <sub>2</sub> (ODAプリンタ)	80 ↓ 9F	N <sub>88</sub> BASIC使用	F1 ↓ FF	ユーザ解放
F	システム予約	A0 ↓ AF	グラフLIO		
10	セントロニクス・プリンタ				
11	拡張バスINT <sub>3</sub> (5" HD)				
12	拡張バスINT <sub>4</sub> (5" FD/2D)				
13	拡張バスINT <sub>4</sub> (8" FD)				



内部割り込みは、8086のアーキテクチャにより使用法が決まっている割り込みと、ソフトウェアによる割り込みとに分類できます。

CPUにより使用法が決まっている割り込み番号は、  
**割り込み番号 0** ……除算のときに 0 で割り算を行った場合に生ずる

**割り込み番号 1** ……シングル・ステップ動作をしているときに生ずる

**割り込み番号 3** ……ブレーク・ポイントのセットに使用する

**割り込み番号 4** ……INTO命令で使用する

の 4 種類です。

また、ソフトウェア割り込みはCPUによる、

INT  $\times\times$  ( $\times\times$ は割り込み番号)

の命令により、その割り込み番号のベクトルが示すアドレスに制御を移します。このときの**割り込み番号は、0 から255までの値をとることができます**から、ソフトウェア割り込みは256種類あることになります。

表2-1がPC9801シリーズでの割り込みベクタ・テーブルの内容です。ただし、このベクタ・テーブルはN<sub>88</sub>BASIC起動の場合の内容であり、MS-DOSなど他のDOSで起動した場合には、割り込み番号0～1Fhまでの内容はほぼ一致していますが、20h～Fhはまったく異なる内容になります。

#### ▶ ハードウェア割り込み

ハードウェア割り込みにも2種類があり、CPUの**NMI端子に直接かかるNMI(Non Maskable Interrupt)**と、**割り込みコントローラ8259Aを通してかかる普通のハードウェア割り込み**があります。

PC9801シリーズでは、NMIはメモリのパリティ・エラーの検出に用いられていますが、I/OポートのNMIマスク用のフリップフロップで割り込み禁止にすることもできます。

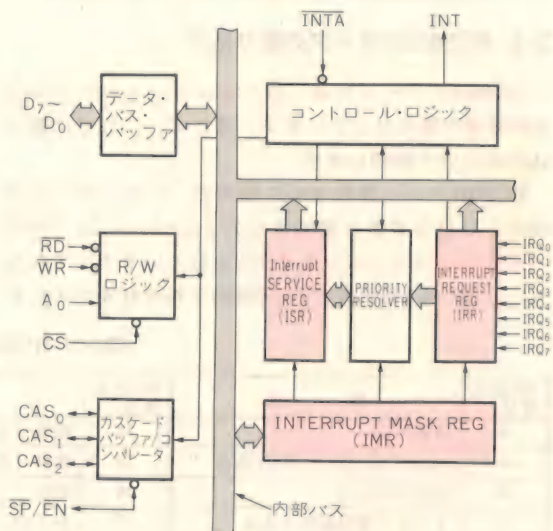
## 2-4 8259Aによるハードウェア割り込み

PC9801シリーズでは、ハードウェア割り込みの制御に、図2-6に示したように2個の8259Aを使用しています。8259Aの内部ブロックは図2-7に示したような構成で、1個で8本の外部割り込みを制御することができ、2個の8259Aにより7本ずつ、計**14本の割り込みを使用**しています。

その内訳は、本体内部のI/Oなどに6本、拡張バスに8本です。PC9801にはBIO(Basic I/O)というソフトウェアによる周辺装置のサポートがありますから、ユーザが割り込み処理のプログラムを作る必要はまずありません。

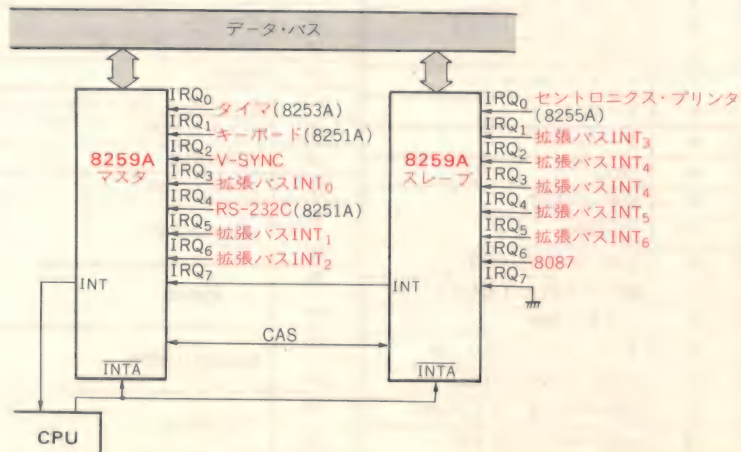
しかし、特別なハードウェアを使用する場合に高速な処理が必要になって、タイミングや時間の管理を正確に行わなければならないような場合には、割り込みコントローラと周辺I/Oデバイスとの関係を知ってい

〈図2-7〉 8259Aの内部ブロック図



〈図2-6〉

2個の割り込みコントローラの接続関係



なければなりません。

8259Aには、IRQ<sub>0</sub>からIRQ<sub>7</sub>までの8本の割り込み要求入力信号線があり、それらに対応する8ビット・レジスタを内部にもっています。

すなわち、

- ① 割り込み要求レジスタ (IRR)
- ② 割り込みサービス・レジスタ (ISR)
- ③ 割り込みマスク・レジスタ (IMR)

の三つです。

これらは、現在の割り込みの状態を知らせるもので、IRRは割り込み処理を要求しているすべての割り込み信号の状態を知ることができ、ISRはその時点で実際に処理している割り込みがどれかを知ることができます。

また、割り込みを禁止するにはCPUによるCLI、STI命令で制御することもできますが、この場合にはすべての割り込みが同時に禁止されてしまいます。そこで、IMRは8個の割り込み要求について独立に割り込みの禁止/許可をすることができるレジスタで、さらに禁止の状態を知ることができます。

割り込み要求が発生すると、IRRの割り込みに対応するビットが立ちます。そして、8259AはCPUに対してINTの信号を送り、CPUの処理を中断させます。8259Aは、そのとき要求されている割り込みのうち、最も優先順位の高い割り込みをISRにセットして、あらかじめセットされている8ビットの0から255までの割り込み番号の一つをCPUに送ります。

CPUは、その割り込み番号に対応する割り込みベ

クトルに書いてあるアドレスを読み込み、そのアドレスへ制御を移します。そのあとIRET命令に出会うと割り込み発生前の処理に戻ります。

## 2-5 PC9801での8259Aの働き

PC9801シリーズに使われている2個の8259Aは、一方がマスタ、他方がスレーブとなり、マスタはI/O空間の00hと02h番地に配置され、スレーブは08hと0Ah番地に配置されています。

また、通常のイニシャライズ状態では、マスタが256個の割り込み番号の08h番から0Fh番の8個を使用し、スレーブが10h番から17h番を使用しています。

例えば、キーボードをたたいた場合には、マスタの8259AのIRQ<sub>1</sub>にキーボードのインターフェースを行っている8251Aからの割り込み信号が入っていますから、8259AはCPUに割り込み番号09h番を送り、CPUはメモリ空間の00024hから00027h番地の4バイトに書かれているアドレスに制御を移します。

したがって、割り込み処理プログラムを独自のものに変更したり、拡張バスの割り込み信号を使用して、ハードウェアの制御をする場合などは、割り込みの処理プログラムを書いて、そのアドレスを対応するベクトルに書き込むことが必要です。

## 2-6 割り込み処理プログラムの書き方

割り込みの処理中に他の割り込みの要求があった場

〈リスト2-1〉 マスタへの割り込みのEOIの送り方

```
label1:                ; 割り込み入口
    push    ax
    push    ...
    ;
    ( 割り込みの処理 )
    ;
    pop     ...
    mov     al,20h ; マスタにEOIを送る
    out     0,al ;
    pop     ax
    iret
```

〈リスト2-3〉 IRRの読み方

マスタの場合	スレーブの場合
mov al,0Ah	mov al,0Ah ;読み込みの要求
out 00h,al	out 08h,al ;
in al,00h	in al,08h ;IRR読み込み

〈リスト2-4〉 ISRの読み方

マスタの場合	スレーブの場合
mov al,08h	mov al,08h ;読み込みの要求
out 00h,al	out 08h,al ;
in al,00h	in al,08h ;ISR読み込み

〈リスト2-2〉 スレーブへの割り込みのEOIの送り方

```
label2:                ; 割り込み入口
    push    ax
    push    ...
    ;
    ( 割り込みの処理 )
    ;
    pop     ...
    mov     al,20h ; スレーブにEOIを送る
    out     08h,al ;
    mov     al,08h ; スレーブのISRを読む
    out     08h,al
    in      al,08h
    cmp     al,0 ; まだ処理があるか?
    jnz     exit2 ; まだ処理が残っている
    ; マスタにはEOIは送らない
    mov     al,20h ; 全処理終了=マスタにEOIを送る
    out     00h,al
exit2:
    pop     ax
    iret
```

〈リスト2-5〉 IMRの操作の仕方

マスタの場合	スレーブの場合
in al,02h	in al,0Ah ;IMR読み込み
and al,xxx	and al,xxx ; 割り込み許可
or al,yyy	or al,yyy ; 割り込み禁止
out 02h,al	out 0Ah,al ;IMR書き込み



合は、そのときに処理している割り込みの優先順位よりも高い場合にはその処理を中断し、優先順位の高い処理を行います。優先順位の低い要求の場合は現在行っている処理が終了するまで待たされます。

通常の優先順位は、マスタのIRQ<sub>0</sub>が最も高く、IRQ<sub>1</sub>、IRQ<sub>2</sub>…の順に低くなり、スレーブのIRQ<sub>7</sub>が最も低い順位となります。このとき問題となるのは、**割り込み処理が終了したことを割り込みコントローラに知らせる必要がある**ということです。

その時点で処理している割り込みコントローラに割り込み処理の終了を知らせなければ、ISRの割り込みレベルに対応するビットはいつまでもクリアされず、したがって、その割り込みよりも優先度の低い割り込み要求は待たされたままになってしまいます。

このため、**通常は割り込み処理の終了時にEOI (End of Interrupt)を8259Aに送らなければなりません**。EOIを8259Aが受けると、そのレベルの割り込み処理は終了し、他の割り込みを受けられるようになります。

マスタにかかった割り込み処理の場合は、マスタにEOIを知らせるだけですが、スレーブにかかった割り込みの場合は、**マスタとスレーブの両方にEOIを送らなくてはなりません**。

スレーブが複数の割り込みを受け付けている場合は、ISRを監視してすべてのスレーブに対する割り込み処理が終わったことを確かめてから、マスタにEOIを送るという手順を踏みます。これらのプログラム例をリ

スト2-1～リスト2-5に示します。

ただし、BASICを使用するときには、BASIC自体が割り込みコントローラを相当に使用していますので、スレーブの外部割り込みを使用している場合に、割り込み処理プログラムがマスタにもEOIを送るとトラブルが発生する場合があります。

そのような場合、マスタには手をつけず、スレーブのみにEOIを送ると解決する場合があります。

## 2-7 8259Aの初期化

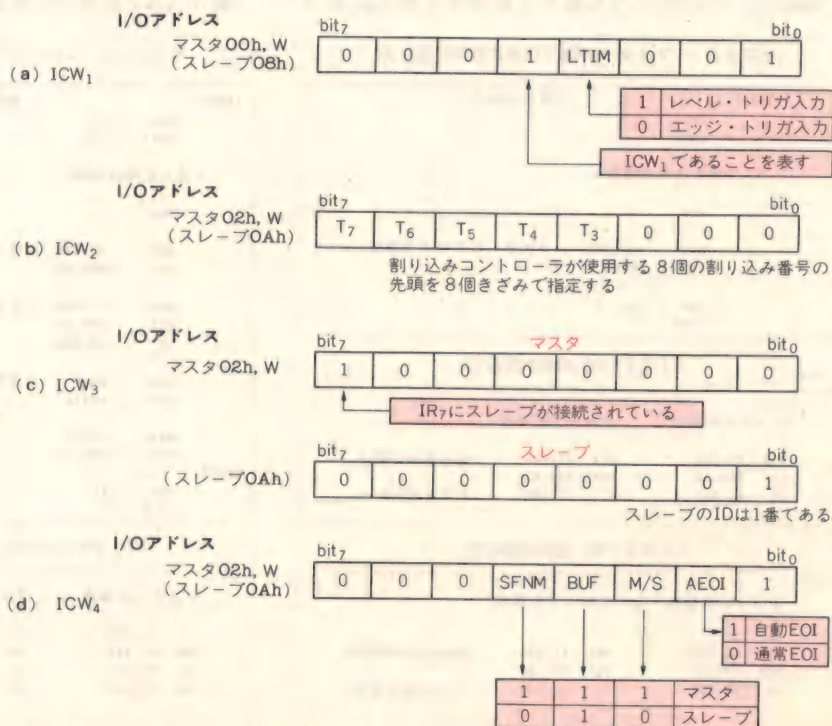
PC9801シリーズに使用している8259Aは**4種類のICW (Initialization Command Words)を設定**しなければなりません。

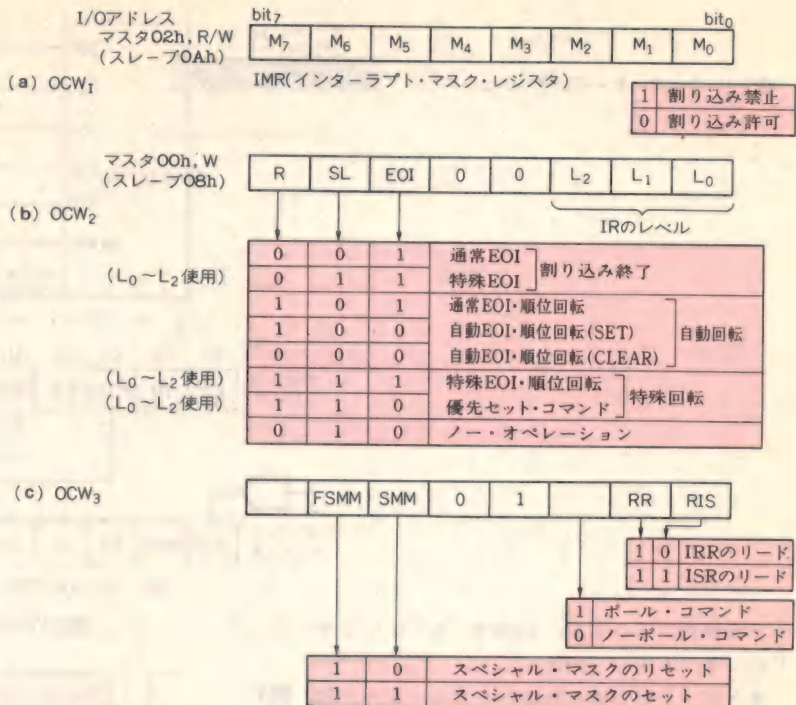
それぞれのICWの意味を図2-8に示します。ICW<sub>1</sub>は、割り込み入力がレベル・トリガかエッジ・トリガかの選択を行います。

また、ICW<sub>2</sub>により、マスタ8259Aは割り込み番号の08h番から0Fh番を使用し、スレーブ8259Aは10h番から17h番を使用するように設定されています。

ICW<sub>4</sub>では、通常のEOIモードと自動EOIモードの設定をします。通常のEOIモードでは、割り込み処理終了時にEOIを発行しないとISRはクリアされませんが、自動EOIモードではCPUに割り込み番号を渡した段階で、ISRを自動的にクリアしてしまいます。したがって、**自動EOIモードでは処理の終わりにEOIを発行する必要がなくなります**。

〈図2-8〉 ICWの意味





以上のICW<sub>1</sub>からICW<sub>4</sub>までの手続きにより、割り込みの受け付けが可能になります。

## 2-8 8259Aのコマンド・ワード

8259Aを初期化した後は、8259Aに対して図2-9に示したような3種類のコマンドOCW(Operation Command Word)を送ることができます。これらの3種のOCWには順序はありません。OCW<sub>1</sub>ではIMRの内容をアクセスすることができ、各割り込み入力に対して割り込み許可と禁止が可能です。

通常は、割り込み入力には優先順位があり、スレープよりはマスタが、IRQ<sub>7</sub>よりはIRQ<sub>0</sub>が高い優先順位をもっています。このため、割り込みを要求すると順位の高いものが優先して処理されますが、割り込みの利用法においては同じ順位で割り込みのサービスを受けたいような場合も発生します。

そのような複雑な要求に対し、優先順位を変更したり、処理が終わった直後の割り込みの順位を下げるといった処理をOCW<sub>2</sub>とOCW<sub>3</sub>により指定することができます。

### [3]PC9801シリーズの周辺ハードウェアとBIOS

PC9801シリーズのほとんどの周辺ハードウェアには、基本的な操作についてBIOSのサポートがあります。これらはおもに内部割り込みの18hから1Chで行われており、ハードウェアを直接操作するソフト

ウェアの使用に当たっては、これらの割り込みペクタを破壊しないのみならず、システム共通領域である00400hから005Fhまでの512バイトのメモリ空間を保存する必要があります。

また、これらのソフトウェアは、使用しているアプリケーションにほとんどよらずに使用することが可能です。以下に各周辺ハードウェアの概要を述べます。

## 3-1 キーボード・インターフェース

キーボード用のコネクタには5本の信号線と、電源、アースの合計7本が出ています。

図3-1に示したようにキーボードはそのうちの6本で本体とつながっています。キーボードの内部にはフロッピー・マイコン8048が用いられており、本体には8251Aが使用されています。8251Aは奇数パリティつき8ビット調歩モードにセットされており、19.2kbpsのシリアル・データをキーボードから受けています。

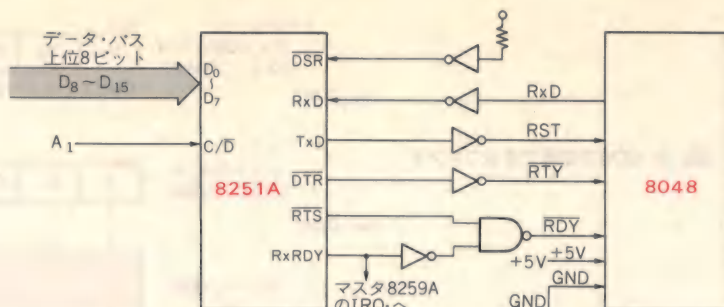
キーボードから本体へは、RxDにキーが押されたときと離れたときに1度ずつ、図3-2に示したようなコードでその情報がシリアル・データで送られてきます。

本体からキーボードへは、RxDとRTSのANDをとったもの、およびTxD、DTRの3本の信号線があります。RTSは、本体がキーボードからの情報を受けることが可能なことを、またDTRは直前に送られた情報の再送の要求をそれぞれ表します。

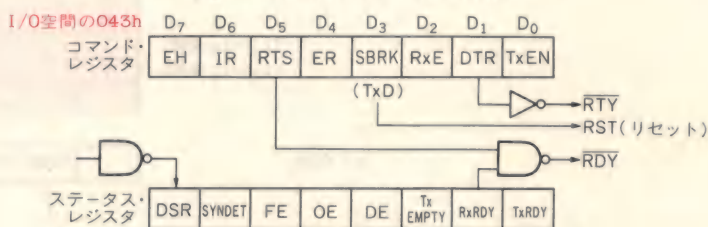
TxDについては、8251Aに対してSend Breakコマ



〈図3-1〉 キーボード・インターフェース



(a) キーボード・インターフェース回路



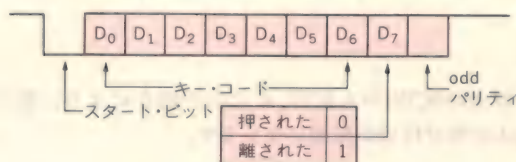
(b) 8251Aの内部レジスタ

ンドを実行することで、TxDを“L”にしてキーボードのリセットをしています。

キーボードから送られるシリアル・データは、図3-3に示したように100個のキーを7ビットで表し、そのキーが押された(0)、離された(1)ことを表すのに1ビットを使用しています。送られる情報はシフト・キー、コントロール・キーなどを含めて物理的なキーの押し/離しの情報で、**キャラクタ・コードとはまったく関係ありません。**

8251AのRxRDYは、割り込みコントローラ8259A

〈図3-2〉 キーボードのシリアル・コード



(マスタ)のIRQ<sub>1</sub>に接続されています。キーボードからのシリアル・データを8251Aが受けると内部割り込み09hが発生します。通常は、割り込み処理プログ

## 5インチ・フロッピー・インターフェースを用いたデータ伝送

PC9801E, Fに標準に装備されている5インチ・フロッピー・ディスク・インターフェースを使用して、2台のPC9801の間でデータを送る簡単なプログラムの例を示します。この部分は、本来はインテリジェント・タイプの5インチ・フロッピー・ディスク・ドライブとのインターフェースのために使用されるものですが、**回路的には8255Aの24本の入出力ピンが直接にコネクタに出ているだけです。**

プログラマブル・ペリフェラル・インターフェース用LSI 8255Aは、最も広く用いられているLSIの一つです。PC9801シリーズでもセントロニクス・プリンタ・インターフェース、5インチ・インテリジェント・フロッピー・インターフェース、システム・ポートと多くの場所に用いられています。また、PC9801シリーズに何らかのハードウェアを接続する場合に使用するチャンスも多いでしょう。

8255Aは、24本の入出力ピンをもっており、12本ずつグループAとグループBとに分けることができます。二つのグループ独立にモードをセットすることが可能です。

▶ **モード0**：単純な入出力ポート

▶ **モード1**：ハンドシェイク入出力ポート

▶ **モード2**：双方向ハンドシェイク・ポート

として使用することが可能です。ただし、モード2はグループAのみが可能です。

このプログラムでは8255Aをもっとも簡単なモード0にインシャライズし直します。送信側のポートAを出力、ポートCの下位4ビットを出力に、上位4ビットを入力にします。受信側はポートAを入力、ポートCの上位4ビットを出力に、下位4ビットを入力にセットして、ポートAでデータの送受を、またポートCでソフトでのハンドシェイクを行っています。

〈図3-3〉 キーのグループ番号

キーの 物理番号 (7bit)	D6	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
D5	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	
D4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1	
D3	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	
D2 D1 D0																		
0 0 0	ESC	8 ユ	( ク	Q タ	O ラ	F ハ	) ム	「 」 ネ	INS	-	6	.		STOP	f・7	SHIFT		bit0
0 0 1	1 ヌ	9 ヨ	) ヨ	W テ	P セ	G キ	Z ツ	> ル	DEL	/	+	NFER		COPY	f・8	CAPS		bit1
0 1 0	2 フ	0 ワ	ヲ イ	E イ	@ 。	H ク	X サ	? メ	↑	7	1			f・1	f・9	カナ		bit2
0 1 1	3 ア	# ア	ニ ホ	R ス	{ 。	J マ	C ソ	- ロ	←	8	2			f・2	f・10	GRPH		bit3
1 0 0	4 ウ	\$ ウ	へ ヘ	T カ	⌋ カ	K ノ	V ヒ	△	→	9	3			f・3		CTRL		bit4
1 0 1	5 エ	% エ	↑ キ	Y ン	A チ	L リ	B コ	XFER	↓	*	=			f・4				bit5
1 1 0	6 オ	& オ	BS	U ナ	S ト	+ レ	N ミ	ROLL UP	HOME CLR	4	0			f・5				bit6
1 1 1	7 ヤ	・ ヤ	TAB	I ニ	D シ	* ケ	M モ	ROLL DOWN	HELP	5	.			f・6				bit7
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		

グループの中の対応するビット

グループの中の対応するビット

ラムがデータを取り込み、キーボードの内部状態を BIOS内部に保持し、さらに**COPY**キーか**STOP**キーが押されたなら内部割り込み05 hか内部割り込み06 hを発生させます(図3-4)。

BIOS内部にはキーボードの情報が常に用意されており、**キーの押し/離しのたびにその情報は更新**されています。ユーザがキーボードの情報を得るには

BIOSを用いるのが最も簡単です。もちろん内部割り込みの09 h番を書き換えて直接データを得てもかまいませんが、キーボードから送られてくるのはキャラクタ・コードではありません。8251AはI/Oアドレスの41 hと43 hにあります。

キーボード操作に関するBIOSは、**内部割り込み18 hによって呼び出します**。キーボードに関する内部

#### 〈リストA〉 2台のPC9801間での送受信プログラム

```

/* 送信プログラム */
#include <stdio.h>
#define EOF (-1)

#define PORTA 0x51 /* 8255の各アドレス */
#define PORTB 0x53
#define PORTC 0x55
#define MODE 0x57

main() {
    int c;
    init_pio();
    while( (c=getchar()) != EOF )
        send( c );
}

init_pio() /* 8255のイニシャライズ */
{
    outportb( MODE, 0x88 ); /* ポートAとポートCの下位を */
    outportb( PORTC, 0 ); /* 出力にイニシャライズ */
}

send( c )
{
    int c;
    while( ( inportb( PORTC ) & 0xF0 ) != 0x00 )
        ; /* 受信側が受信可能になるのを待つ */
    outportb( PORTA, c ); /* 一文字送信 */
    outportb( PORTC, 0xF0 ); /* 送信したことを知らせる */
    while( ( inportb( PORTC ) & 0xF0 ) != 0xF0 )
        ; /* 受信側が受信するのを待つ */
    outportb( PORTC, 0 ); /* 送信終了を送る */
}

```

```

/* 受信プログラム */
#include <stdio.h>
#define EOF 0x1A

#define PORTA 0x51 /* 8255の各アドレス */
#define PORTB 0x53
#define PORTC 0x55
#define MODE 0x57

main() {
    int c;
    init_pio();
    while( (c=recchar()) != EOF )
        putchar(c);
}

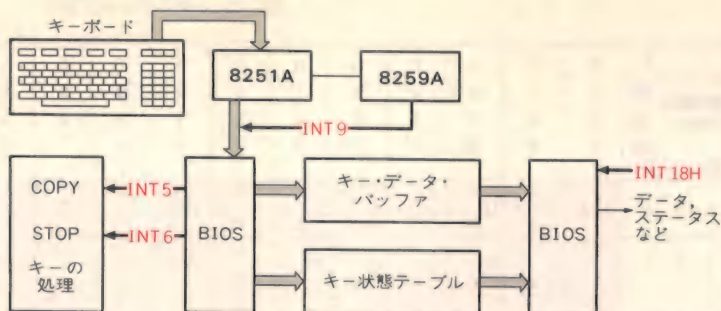
init_pio() /* 8255のイニシャライズ */
{
    outportb( MODE, 0x93 ); /* ポートAとポートCの下位を */
    outportb( PORTC, 0 ); /* 入力にイニシャライズ */
}

recchar()
{
    while( ( inportb( PORTC ) & 0xF0 ) != 0xF0 )
        ; /* 送信されるのを待つ */
    c = inportb( PORTA ); /* 一文字受信 */
    outportb( PORTC, 0xF0 ); /* 受信したことを知らせる */
    while( ( inportb( PORTC ) & 0xF0 ) != 0xF0 )
        ; /* 送信側の送信終了を待つ */
    outportb( PORTC, 0 ); /* 受信終了を送る */
    return c;
}

```



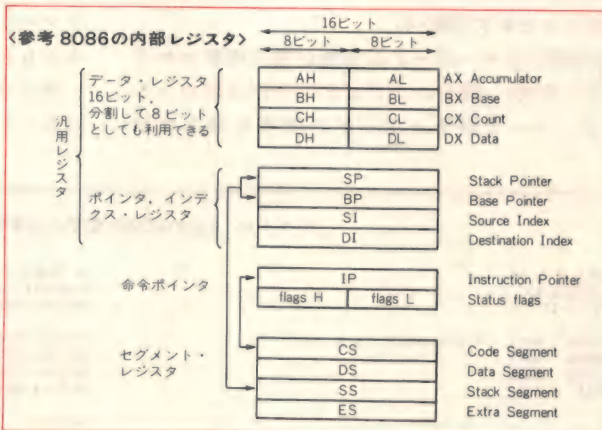
〈図3-4〉  
キーボードからの入力データの流れ



〈図3-5〉  
キーボードに関するBIOSのサポート  
(内部割り込み18h)

AH	動作	リターン・バリュ
00h	1文字入力 キー・バッファが空なら入力を待つ	AH=キー・トップに1対1対応の物理番号 AL=キャラクタ・コード
01h	キー・バッファの先頭を見る。 バッファは変化しない。	BH=0:バッファが空, 1:中身がある AH=キー・トップに1対1対応の物理番号 AL=キャラクタ・コード
02h	シフト・キーの状態	AL=押されているキーに対応するビットが"1"になる。 <div style="display: flex; justify-content: space-around; align-items: center;"> <span>D<sub>4</sub></span> <span>D<sub>3</sub></span> <span>D<sub>2</sub></span> <span>D<sub>1</sub></span> <span>D<sub>0</sub></span> </div> <div style="display: flex; justify-content: space-around; align-items: center;"> <span>CTRL</span> <span>GRPH</span> <span>カナ</span> <span>CAPS</span> <span>SHIFT</span> </div>
03h	キーボード関係の初期化	バッファ、ハードウェアの初期化
04h	キーボードの状態のセンス	アークギュメント AL=グループ番号 (00h~0Fh) リターン・バッファ AH=グループのキーの状態

100個のキーを8個ずつ16組のグループにわけ、そのグループのキーの状態を得る



割り込みを図3-5に示します。呼び出した後は、出力以外のすべてのレジスタは保存されます。

BASICからはINP関数を用いて、I/O空間の00hから0Chまでをスキャンすることで、キーボードの状態を直接に知ることができます。これは、PC8000シリーズとの互換性のためにBASIC上でソフトウェアがサポートしているもので、I/Oの00hから0Chに物理的に何かのチップが繋がっているわけではありません。

したがって、CPUからIN命令を用いてこれらのアドレスを読んでも、キーボードの情報は何も得られません。実際、今ブームのバンク切り替え方式のキャ

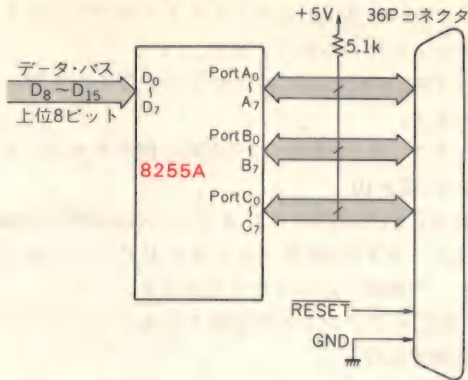
ッシュRAMボードは、I/Oの0Chを用いたものが多いようです。

キーボード用の8251Aは受信のみに用いられていますが、Tx/Dもコネクタに出ていますので、キーボードに本体からのデータを受信する能力があれば、キーボードにコマンドなどを送ることもできます。実際にキーボードのコネクタに接続して、本体からのデータを表示できるような装置も市販されているようです。

### 3-2 5インチ・フロッピー・ディスク ・インターフェース

これは、汎用I/Oインターフェースとして最も代表

〈図3-6〉 5 インチ・インテリジェント・フロッピー・ディスク・インターフェース



〈図3-8〉 プリンタに関するBIOSのサポート  
(内部割り込み1Ah)

AH	動作	パラメータ
10h	セントロニクス・プリンタ初期化	AH は破壊
11h	1 バイト印字出力	AL = 印字文字 AH = bit 0 0 : 未出力 bit 1 1 : タイム・アウト
12h	プリンタ・ステータス	AH = 0 : 出力できない 1 : 出力可能
30h	文字列出力	ES: [BX] → バッファのポインタ, CX: データ長 AH = 00h 正常終了

的な回路です。試作回路を、一時的にPC9801に接続する場合などに流用することもできます。このインターフェースはPC9801、E、Fには標準装備でしたが、VM、VFからは外されてしまいました。

このインターフェースは、8255Aを使用しており、A、B、Cの3つのポートが5kΩでプルアップされてコネクタに直接出ています(図3-6)。8255AのI/Oアドレスは51h、53h、55h、57hです。

通常は、イニシャライズにより、モード0でCポートの下位4ビットとAポートが入力、Cポートの上位4ビットとBポートが出力にイニシャライズされています。

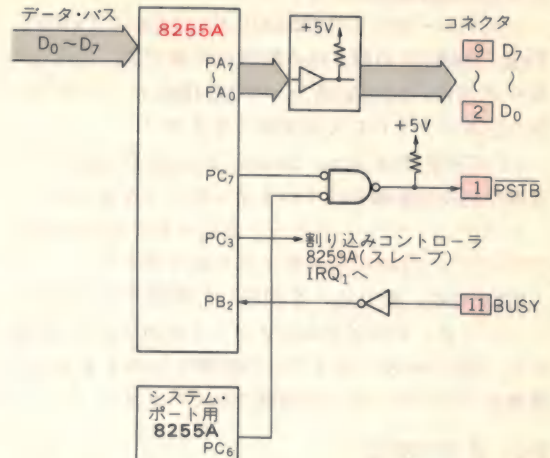
汎用インターフェースとして流用する場合は、出力同士がぶつからないようにイニシャライズし直してから、外部機器を接続する必要があります。

### 3-3 セントロニクス・プリンタ ・インターフェース

セントロニクス準拠の8ビット・パラレル・プリンタ用インターフェースのために、8255Aが標準装備されています(図3-7)。プロッタなど各種の周辺装置用のパラレル・インターフェースとしてセントロニクス・インターフェースは広く用いられています。

PC9801シリーズに実装されているプリンタのコネクタは、標準のものよりかなり小さく、データ8本とBusy、Strobeの必要最小限の信号線しかつながっ

〈図3-7〉 セントロニクス・プリンタ・インターフェース



ていないので、紙切れやセレクトなどプリンタの情報はいつさいわかりません。

システム・ポート用8255AのポートCのビット6にプリンタ・ストローブをマスクするビットがあります。

セントロニクス用8255AのポートCのビット7は、8259A(スレプ)のIRQ<sub>0</sub>に接続されています。一見、8255Aのモード1によってハンドシェイクできそうに思えますが、アクノレッジがつかないようです。いったいこの割り込みをどのように使用しているかは不明です。

プリンタ用8255AのポートBには、プリンタの情報以外にシステムの情報が各種見えます。この情報はとても有効ですが、PC9801E、FとVM、VFでは内容がかなり違いますので、機種に依存しないプログラムを作る場合には注意が必要です。

プリンタに関するソフトウェアのサポートを図3-8に示します。

### 3-4 RS-232Cインターフェース

RS-232Cのインターフェースは、標準的な8251Aを使用した回路です。ボーレートはタイマ8253Aのチャネル2によって作られるクロックと、8251Aの内部の分周比によって決まります。

8253Aのチャネル2はモード3にセットされており、分周比2から65535まで指定可能ですから、8251Aには1228.8kHzから37.5Hzまでのクロックが供給可能



です。これにより、1ボー以下から、19.2kボーまでの任意の伝送速度が得られます。

システム・ポートの8255AのポートCからTxRDY, TxR, RxRDYの割り込み制御が可能です。これらの割り込みは、8259A(スレーブ)のIRQ<sub>4</sub>につながり、割り込みベクタ0Chが使用できます。

SYNDET/BD(Sync Detect/Break Detect)による割り込みは配線されていませんのでできません。

システム・ポートのポートBからRS-232CのCI, CTS(CS), CDの状態を得ることができます。

BIOSには、割り込みを使用した受信データのバッファリング・プログラムのサポートがあります。すなわち、図3-9に示したように内部割り込み19hに7種類のファンクションが用意されています。

3-5 タイマ8253

前にも述べたように、PC9801にはプログラマブル・インターバル・タイマ8253が使用されています。8253は3個のカウンを内部にもっています。

カウンタ#0は出力が8259A(マスタ)のIRQ<sub>0</sub>に接続

されていますので、内部割り込み08hを発生することができます。

カウンタ#1はDMACのチャンネル1と共にメモリ・リフレッシュに用いられています。

カウンタ#2はRS-232Cのクロックを作るのに用いられています。

また、カウンタ#0を用いたBIOSの便利なサポートがあります(図3-10)。

すなわち、CPUのAHに02h, CXに時間(10ms単位), ES:[BX]に処理プログラムのアドレスをセットして、内部割り込み1Chを行うと、

①ES:[BX]のアドレスを内部割り込み07hのベクトルに書き込む

②カウンタ#0を10msにセット

③AXを壊してリターン

という動作をします。

このあと、カウンタ#0によりCXで指定した回数の割り込み08hが起こるとBIOSは内部割り込み07hを起こし、先に指定した処理プログラムが実行されます。

〈図3-9〉 RS-232Cに関するBIOSのサポート(内部割り込み19h)

AH		動作		パラメータ																								
00h	RS-232Cの初期化 X <sub>on</sub> , X <sub>off</sub> を行わない	AL=ボーレート	<table><tr><td>0</td><td>75ボー</td><td>4</td><td>1200ボー</td></tr><tr><td>1</td><td>150ボー</td><td>5</td><td>2400ボー</td></tr><tr><td>2</td><td>300ボー</td><td>6</td><td>4800ボー</td></tr><tr><td>3</td><td>600ボー</td><td>7</td><td>9600ボー</td></tr></table>				0	75ボー	4	1200ボー	1	150ボー	5	2400ボー	2	300ボー	6	4800ボー	3	600ボー	7	9600ボー	ES:[DI]→INT					
			0	75ボー	4	1200ボー																						
1	150ボー	5	2400ボー																									
2	300ボー	6	4800ボー																									
3	600ボー	7	9600ボー																									
01h	RS-232Cの初期化 X <sub>on</sub> , X <sub>off</sub> を行う	CH=8251Aのモード設定 D <sub>0</sub> <table><tr><td>S<sub>2</sub></td><td>S<sub>1</sub></td><td>EP</td><td>PEN</td><td>L<sub>2</sub></td><td>L<sub>1</sub></td><td>B<sub>2</sub></td><td>B<sub>1</sub></td></tr></table>	S <sub>2</sub>	S <sub>1</sub>	EP	PEN	L <sub>2</sub>	L <sub>1</sub>	B <sub>2</sub>	B <sub>1</sub>	+02h	内部フラグ		8251Aコマンド														
			S <sub>2</sub>	S <sub>1</sub>	EP	PEN	L <sub>2</sub>	L <sub>1</sub>	B <sub>2</sub>	B <sub>1</sub>																		
			+04h	タイム・アウト時間																								
			+06h	X <sub>off</sub> 発生データ長																								
			+08h	X <sub>on</sub> 発生データ長																								
+0Ah	バッファの先頭アドレス																											
	▶タイマ8253Aのカウンタ#2のセット ▶8251Aのモードコマンドのセット ▶受信バッファのセット	CL=8251Aのコマンド・ワード D <sub>0</sub> <table><tr><td>EH</td><td>IR</td><td>RTS</td><td>ER</td><td>SBRK</td><td>RxE</td><td>DTR</td><td>TxE</td></tr></table>	EH	IR	RTS	ER	SBRK	RxE	DTR	TxE	+0Ch	バッファの終わり+1																
			EH	IR	RTS	ER	SBRK	RxE	DTR	TxE																		
			+0Eh	バッファの有効長																								
			+10h	バッファの空エリアの先頭																								
			+12h	バッファの有効データの先頭																								
+14h	受信データ	ステータス																										

〈図3-10〉 タイマ、カレンダーに関するBIOSのサポート(内部割り込み1Ch)

AH	動 作	パラメータ	リターンバリュ												
00h	日付け,時刻の読み出し	6バイトのバッファへのポインタ	AXは破壊												
01h	日付け,時刻の設定	ES:[BX]→ <table><tr><td colspan="2">年(BCD)</td></tr><tr><td>+1</td><td>月 曜日</td></tr><tr><td>+2</td><td>日(BCD)</td></tr><tr><td>+3</td><td>時(BCD)</td></tr><tr><td>+4</td><td>分(BCD)</td></tr><tr><td>+5</td><td>秒(BCD)</td></tr></table>	年(BCD)		+1	月 曜日	+2	日(BCD)	+3	時(BCD)	+4	分(BCD)	+5	秒(BCD)	
年(BCD)															
+1	月 曜日														
+2	日(BCD)														
+3	時(BCD)														
+4	分(BCD)														
+5	秒(BCD)														
02h	BIOS内部のインターバル・タイマの起動	CX=インターバルの時間(10ms単位) ES:[BX]=タイム・アウト時の割り込み処理エントリ	AXは破壊												

〈リスト3-1〉 インターバル・タイマの起動プログラム例(10分間は処理を続けるが、その後にはコールドBOOTする)

```

mov ax,0FD80h
mov es,ax
mov bx,0
mov cx,0EA60h
mov ax,2
int 1Ch

```

そのプログラム例をリスト3-1に示します。

### 3-6 カレンダー

PC9801シリーズはカレンダー時計μPD1990Aをもち、バッテリーによってバックアップがされています。μPD1990Aを直接アクセスするのは面倒ですが、幸いBIOSのサポートがあります。

AH=00h, ES:[BX]=6バイトのバッファへのポインタをセットして内部割り込み1Chを行うと、図3-10に示した年、月、曜日、日、分、秒の情報が得られます。このときAXは壊されます。

逆に、AH=01h, ES:[BX]=6バイトのバッファへのポインタをセットして、内部割り込み1Chを行うとμPD1990Aがセットされます。このときもAXは壊されます。

### 3-7 システム・ポート

システム・ポートには8255Aが使用されており、デバッグ・スイッチの状態を読み込んだり、他のインターフェースのサポートをしています。

ポートAは入力で、デバッグ・スイッチSW<sub>2</sub>につながっています。

ポートBは入力で、各種情報を読み込みます。

ポートCは出力で、RS-232Cの割り込み、ブザー、プリンタ・ストローブの制御を行っています。I/Oポートの一覧表などを参考にしてください。

また、次のようなブザーについてのBIOコマンドがあります。

▶AH=17hにセットして内部割り込み18hを行うとブザーを鳴らす。

▶AH=18hにセットして内部割り込み18hを行うとブザーを止める。

### 3-8 CRTディスプレイ

PC9801シリーズには、とてもパワフルなグラフィック機能があります。640×400ドットの画面を白黒で6枚、カラーで2枚、640×200ドットであればその倍

の枚数をもっています。VM、VFからは、画面の合成やアナログRGBによるパレット操作なども可能になりました。

μPD7220は、マスタとスレーブの2個が用いられています。マスタはテキスト画面、カーソルなどの出力、同期信号の発生などを行っています。スレーブはマスタに同期して、グラフィック専用に使われます。そのほかに、μPD52611 CRT M/S(マスタ・スライス)をもっており、スムーズ・スクロールも可能です。

BIOSのサポートは、テキスト画面については便利なサポートがありますが、グラフィックスについては複雑な操作が必要です。内部割り込み18hのAH=0Ahから15hまでがテキスト画面に関するもの(図3-11)、AH=40hから4Ahまでがグラフィック画面に関するものです(図3-12)。特に、AH=45hから49hまでは、グラフィック操作についてUCWと呼ばれる作業領域を使用します。

BASICのROMの上には、LIOと呼ばれるもっと高レベルのグラフィック用のファンクションが用意されており、幸いなことにBASIC以外のアプリケーションからもその機能を使用することができます。ただし、BASIC以外では割り込みベクタがセットされておらず、作業領域もないため、機種に依存しないプログラムを書くためにも使用する前にいくつかの準備が必要です。それを簡単に説明します。

① 割り込みベクタの内容をセットしなければならない。

F9900h番地にあるエントリ・テーブルからエントリ・ポイントを読んで割り込みベクタにセットしなければなりません。エントリ・テーブルの形式は図3-13のようになっており、割り込み番号とエントリ・ポイントのオフセットの組です。各エントリのセグメントはF9900hです。

割り込みベクタをセットするプログラムの例をリスト3-2に示します。

② ワーク・エリアを用意しなければならない。

データ・セグメントのオフセット0hから1200



〈図3-11〉 テキスト画面に対するBIOSのサポート (内部割り込み18h)

AH	動作	パラメータ																
			D3		D2	D1	D0											
0Ah	CRTモードの設定	AL⇐モード情報	0 0 0 0		KCG 0:コード・アクセス 1:ドット・アクセス	アトリビュート 0:バーチャル・ライン 1:簡易グラフ	桁数 0:80字 1:40字	行数 0:25行 1:20行										
0Bh	CRTモードのセンス	AL⇒モード情報	0:標準 1:高解像	0 0 0	同上	同上	同上	同上										
0Ch	テキスト画面表示開始																	
0Dh	テキスト画面表示停止																	
0Eh	テキスト画面の一つのVRAMアドレス指定	DX=表示開始アドレス (ただし16進5桁目はAに固定)																
0Fh	複数の表示領域を指定	<div><div>BX:[CX]→</div><div><div>4バイト</div><table><tr><td>VRAMのアドレス</td><td>表示行数</td></tr><tr><td>+4</td><td>VRAMのアドレス</td></tr><tr><td>+8</td><td>⋮</td></tr><tr><td>+ (4×N-4)</td><td>VRAMのアドレス</td></tr><tr><td></td><td>表示行数</td></tr></table></div></div> <div>DL = エントリの数 (N) (1~4) DH = 表示をはじめるエントリの番号 [(0~(N-1))]</div>							VRAMのアドレス	表示行数	+4	VRAMのアドレス	+8	⋮	+ (4×N-4)	VRAMのアドレス		表示行数
VRAMのアドレス	表示行数																	
+4	VRAMのアドレス																	
+8	⋮																	
+ (4×N-4)	VRAMのアドレス																	
	表示行数																	
10h	カーソルのタイプ	AL=00:ブリンクする      01:ブリンクしない																
11h	カーソルの表示																	
12h	カーソルの停止																	
13h	カーソルの位置	DX=VRAMアドレス																
14h	フォント・パターン読み出し	BX:[CX]=パターンを展開するバッファ・アドレス (2バイト+8,16,32バイト) DX=展開するコード (ANKの場合はDH=00h)																
(15h)	ライト・ペン位置	AH: 0なら押されている,    1なら押されていない DX: ペン位置のテキストVRAMアドレス																
16h	テキストVRAM初期化	DH=初期化アトリビュート DL=初期化キャラクタ																
19h	ライト・ペン初期化	AH破壊																
1Ah	ユーザ文字定義	BX:[CX]=フォント・パターン・バッファへのポインタ DX=登録コード																
1Bh	KCGアクセス・モード	AL=0:コード・アクセス      1:ドット・アクセス																

〈図3-12〉  
グラフィックに関するBIOSのサポート  
(内部割り込み18h)

AH	動作	パラメータ																																												
40h	グラフィック表示開始																																													
41h	表示停止																																													
42h		<div>CH = <table><tr><td colspan="2">D7</td><td colspan="2">D6</td><td colspan="2">D5</td><td colspan="2">D4</td><td colspan="2">D3</td><td colspan="2">D2</td><td colspan="2">D1</td><td colspan="2">D0</td></tr><tr><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">1</td></tr></table></div> <div><div>→ 表示バンク指定 (0 or 1) 16000</div><div>→ 0: カラー 1: 白黒</div><div>→ UPPER } VRAM領域指定 32000</div><div>→ LOWER } (10進)</div><div>VRAM 0 LOWER UPPER</div></div>	D7		D6		D5		D4		D3		D2		D1		D0		0		1		0		1		0		1		0		1													
D7		D6		D5		D4		D3		D2		D1		D0																																
0		1		0		1		0		1		0		1																																
43h	パレット・レジスタのセット	DS: [BX] = UCWへのポインタ																																												
44h	ボーダ・カラーのセット																																													
45h	ドットの書き込み	<div>CH = <table><tr><td colspan="2">D7</td><td colspan="2">D6</td><td colspan="2">D5</td><td colspan="2">D4</td><td colspan="2">D3</td><td colspan="2">D2</td><td colspan="2">D1</td><td colspan="2">D0</td></tr><tr><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">1</td></tr></table></div> <div><div>↑ 0: 標準</div><div>↑ 0: LOWER ALL</div><div>↑ 1: 高解像 1: UPPER</div><div><table><tr><td>0</td><td>0</td><td>P1 (P4)</td></tr><tr><td>0</td><td>1</td><td>P2 (P5)</td></tr><tr><td>1</td><td>0</td><td>P3 (P6)</td></tr><tr><td>1</td><td>1</td><td>P1 ~ P3 (P4 ~ P6)</td></tr></table></div></div>	D7		D6		D5		D4		D3		D2		D1		D0		0		1		0		1		0		1		0		1		0	0	P1 (P4)	0	1	P2 (P5)	1	0	P3 (P6)	1	1	P1 ~ P3 (P4 ~ P6)
D7		D6		D5		D4		D3		D2		D1		D0																																
0		1		0		1		0		1		0		1																																
0	0	P1 (P4)																																												
0	1	P2 (P5)																																												
1	0	P3 (P6)																																												
1	1	P1 ~ P3 (P4 ~ P6)																																												
46h	ドットの読み込み																																													
47h	直線・方形の書き込み																																													
48h	円弧の書き込み	DS: [BX] = UCWへのポインタ																																												
49h	グラフィック文字の書き込み																																													
4Ah	描画タイミング・モード	CH = 06H フラッシュ描画 (高速書き込みモード) 16H フラッシュレス描画																																												

(注) UCWについては膨大な知識が必要

〈図3-13〉 エントリ・テーブルの形式

アドレス	+0	+1	+2	+3
F9900h	エントリの 個数N			
F9904h	内部割り込み 番号1	0	エントリの オフセット・アドレス1	
F9908h	内部割り込み 番号2	0	オフセット・アドレス2	
	⋮			
	内部割り込み 番号N	0	オフセット・アドレスN	

hバイト(GCOPY-内部割り込みC5hを使う場合は1400hバイト)のワーク・エリアを用意しなければなりません。

③ 128バイト以上のスタック・エリアを用意しなければならない。

④ 内部割り込みC5hのベクタをセットしなければならない。

時間のかかる処理を途中で外部から中断できるように、処理をしている間は一定時間おきに内部割り込みC5hが発生します。このため、少なくともC5hのベクタはIRETへのポインタでなければなりません。できれば、STOPキーのチェックなどもしてください。

以上の準備を行って初期化を行い、はじめて使用可能になります。

#### 4 PC9801VXについて

1986年10月にPC9801シリーズの新製品が発表されました。V30に加えて、80286が実装されたPC9801

〈リスト3-2〉 エントリ・ベクタのセット・プログラム

```

/* エントリベクタのセットプログラム */
#define SEG_VECT ((unsigned) 0)
#define SEG_GLIO ((unsigned) 0xF990) /*セグメントアドレス*/

set_vector()
{
    unsigned i, j;
    unsigned intno;

    for( i=4, j= 0xFF & peek(0,SEG_GLIO); j--; i+=4 ) {
        intno = 0xFF & peek(i,SEG_GLIO);
        pokew( intno*4+2, SEG_VECT, SEG_GLIO );
        pokew( intno*4, SEG_VECT, peek(i+2,SEG_GLIO) );
    }
}

```

VXシリーズとXLシリーズが登場したことが最大の話題です。

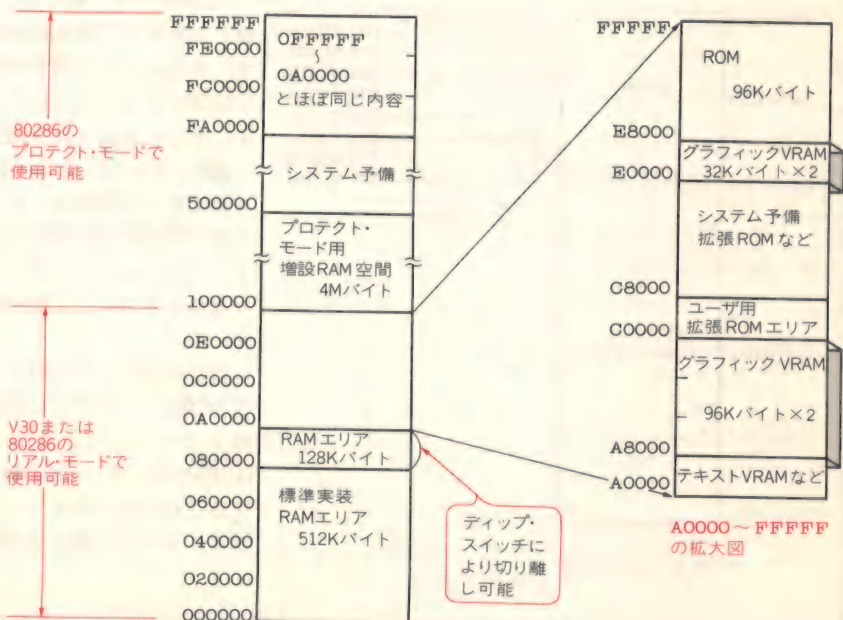
これは、1台のパーソナル・コンピュータの中に2種類のシステムが存在するようなもので、これらのシリーズも過渡期の製品であることは間違いないところ です。

同時にPC9801LTというラップトップ型の携帯が可能な製品も発表されました。将来は、このような小型システムとある程度規模の大きなシステムとが明確に区別されるはず です。

ここでは、VXシリーズについて従来機種との違いを中心に説明します。

このシリーズは、従来のVMシリーズに合わせて、1Mバイト・フロッピー・ディスク・インターフェースのみのVX0、5インチ1Mバイト・フロッピー・ディスク装置2台を内蔵したVX2、5インチ1Mバイト・フロッピー・ディスク装置2台と3.5インチ20Mバイト・ハード・ディスク装置を内蔵したVM4の3種類があります。

〈図4-1〉 PC9801VXのメモリ空間





その特徴を簡単に述べると、

- ① V30(クロックは 8 MHz/10MHzが切り替え可能)と 80286(クロック 8 MHz)の 2 種類のCPUを実装。
- ② 内部RAMは、640Kバイトを標準で実装しており、従来からの市販のRAMディスクが利用可能なように 80000 h 番地からの128Kバイトは切り離しが可能。
- ③ グラフィック用V-RAMは、128Kバイトが 2 画面あり、16色表示が可能。
- ④ グラフィック表示能力が強化されている。
- ⑤ 漢字ROMは、JIS第 2 水準まで標準装備。
- ⑥ 80286のプロテクト・モードでは 4 MバイトまでRAMボードを増設可能。

80286は、リアル・モードとプロテクト・モードの二つのモードをもっています。リセット直後はリアル・モードで動作していますが、リアル・モードでは8086や80186と上位互換性があり、メモリ空間も8086と同様に 1 Mバイトを使用します。

リアル・モードで動作中に、CPUのMSW(マシン・ステータス・ワード)のPE(プロテクション・イネーブル・ビット)をセットすると、80286はプロテクト・モ

ードに入ります。このモードでは、1 Gバイトの仮想メモリがサポートされ、メモリ空間の保護機構などもサポートされます(図4-1)。

#### 4-1 V30と80286の互換性について

VXシリーズでは、本体前面のディップ・スイッチにより、CPUをV30か80286かを切り替えることができます。V30動作時にはVMなどと同様に、スイッチによってクロック周波数を 8 MHzか10MHzに切り替えることが可能ですが、80286動作時にはスイッチの位置に関係なくクロックは 8 MHzの動作になります。

V30動作時には、VMなどと完全な互換性がありますが、80286のリアル・モード動作時には次のような互換性に対する問題があります。

① 80286は動作が高速であり、VXがハードウェア的にメモリのウェイト・サイクルが短いため、I/O周辺LSIに対するアクセスの間隔が短くなって、LSIによっては誤動作をする場合などがあります。

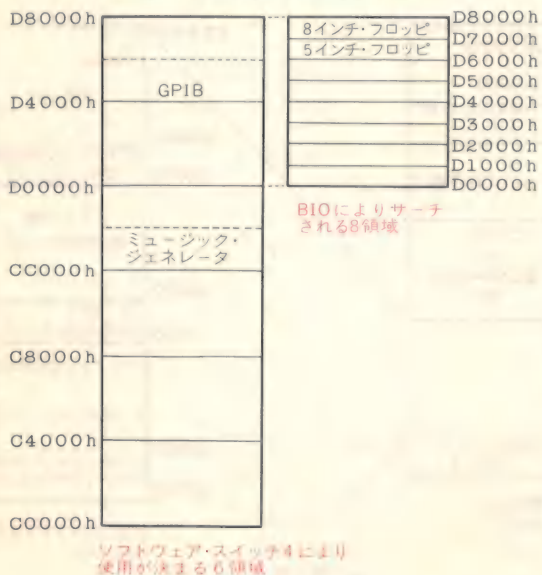
特に、8251Aのアクセスなどがよく問題になります。ソフトウェアによる遅延が影響するような箇所には、十分なタイミングの余裕が必要になります。

## ROMによるオート・スタート

PC9801を制御用を使用する場合などには、電源投入と同時に、フロッピー・ディスクやハード・ディスクなどは使用しないで、自動的にプログラムを動

〈図A〉 拡張ROM領域のメモリ・マップ

1ブロック16Kバイト 1ブロック4Kバイト



かしたい場合が出てきます。ここでは拡張ボード上にユーザが用意したROMからPC9801を起動させる方法を紹介します。

PC9801のリセット直後には、DMA、割り込み、CRT、キーボードなど多くの周辺装置の初期化をしなければなりません。ユーザとしては初期化がすんだ後に制御を受け取ったほうが便利です。

PC9801には、新たなハードウェアを追加した場合などのために、機能拡張の各種の工夫がなされています。その中で、拡張ROM領域とよばれる領域がメモリ空間のC0000 hからD7FFF h番地にあります。そこに装備されるROMには 2 種類があります。

図Aに示したように、D0000 hからD7FFF h番地までの領域は、1000 h = 4 Kバイト単位の 8 個の拡張ROM領域となっています。8 インチや 5 インチのフロッピー・ディスク・インターフェース・ボードの上には、この領域に見えるROMがのっています。

C0000 hからD7FFF h番地までは4000 h = 16Kバイト単位に区切られた 6 個の拡張ROM領域となっていて、ミュージック・ジェネレータやGPIBのボード上のROMは、この領域のものです。

D0000 hからD7FFF h番地までの 8 個の領域は、リセット後の初期化の途中でROMがあるかど

②80286は、PUSH SP命令ではV30や8086と異なる値をスタックへプッシュします。PUSH SPでプッシュした値を参照するようなプログラムは互換性がありません。

③ローテート命令(RCL, RCR, ROL, ROR), シフト命令(SHL, SHR)では、CLレジスタの内容によりシフト数を与えます。このとき、8086では8ビット全部が有効なのですが、80286では下位5ビットのみが有効です。そのため、31以上の値を与えた場合のシフトやローテートの動作には互換性はありません。

④ステータス・レジスタに追加されたビットがありますから注意が必要です。

以上が重要なものですが、その他にV30固有の命令の使用や、プリフィックス命令の異常な使用などが非互換の要因となります。

## 4-2 拡張スロットとアドレス・バス

V30や80286のリアル・モードでは、CPUのアクセスするメモリ空間は1Mバイトですので、拡張バスのアドレスはAB<sub>001</sub>からAB<sub>191</sub>の20本が使用されます。と

ころが、80286のプロテクト・モードでは16Mバイトのメモリ空間が有効となり、24本のアドレスが使用されます。

拡張スロットに装着されるボードには、VMなどに使用されていた20本のアドレスのみをデコードして、AB<sub>201</sub>からAB<sub>231</sub>の上位4ビットを無視しているボードと、24本すべてをデコードしているXLやVX対応のボードとがあります。当然、80286のプロテクト・モードによる16Mバイトのメモリ空間を有効に利用したい場合には、上位4ビットのデコードが必要になります。しかも、デコードされていないボードとの混用を可能にしないではありません。VXの拡張バスには、マイクロ・スイッチとジャンパ・ピンが各スロットに装備されており、上位4ビットのアドレス・デコードの問題を解決しています。

スロットには二つのモードがあります。一つは24ビット・フルデコードしている基板用のモードでAB<sub>001</sub>からAB<sub>231</sub>がそのまま出力されます。もう一つは、20ビット・デコードしている基板用のモードです。24ビット・デコードしている基板には小さなバーが付いており、スロットに装着するとマイクロ・スイッチをバ

うかサーチされます。ですから、D0000hからD7FFFh番地の中のあいているところにユーザがROMを用意しておけば、本体のROMのBIOSから制御をとることができます。

セグメントがD0000hからD7000hまで100hごとに、オフセットの09h番地が

55hで0Ah番地がAAhであるか

を見て、そうであればオフセットの06h番地へFAR CALLしてきます。したがって、そこからユーザのプログラムを書いておけば、自動スタートが可能になります。ROMの先頭から06hのところにジャンプ命令を書いて、09hに55h、0AhにAAhと書いて、そのROMをD0000h、D1000h、D2000h、……D7000hの中のあいているところへ配置しておけばよいわけです。

C0000h番地から16Kバイトごとの6個の拡張ROM領域のほうは、図Bに示すメモリ・スイッチの4番により拡張ROMが存在するかどうかが決まり、その情報によってアドレスの先頭に、BASICの初期化のときに制

御が渡ってきます。

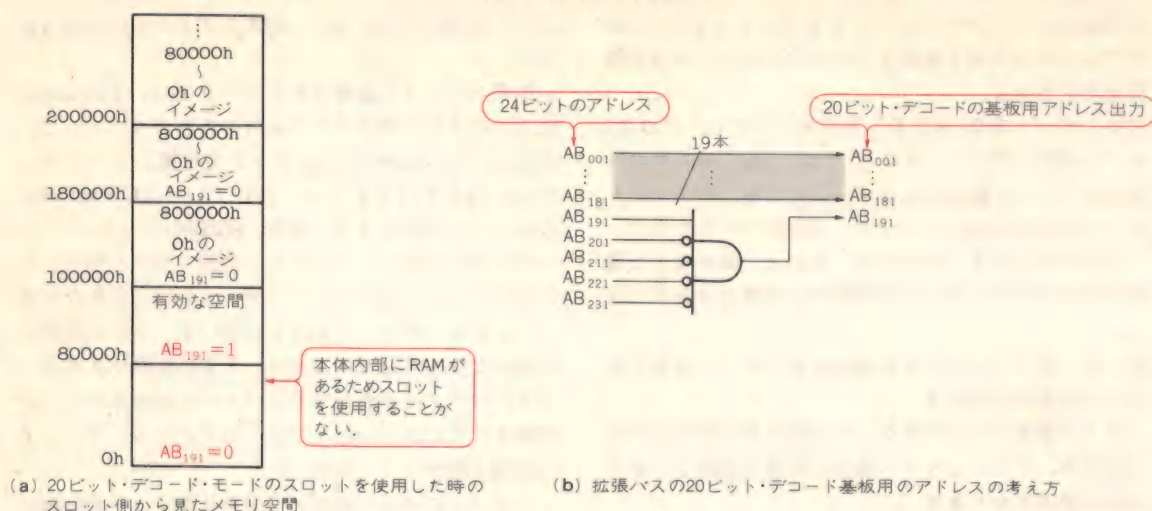
この領域のROMは、通常はBASICの命令や関数を拡張するために用いられていますが、BASICの初期化のときに制御が渡ることを利用して、ユーザのROM上のプログラムを起動させることができます。

〈図B〉 ソフトウェア・スイッチと拡張ROMの関係

論理スイッチ名	メモリ番地	データ(ビット位置)								機能	
		7	6	5	4	3	2	1	0		
SW <sub>1</sub>	A3FEE								0	0でなければならない	
									0	0でなければならない	
								0		拡張ROM接続 C8000~ C9FFFF	なし
								1			あり
						0				拡張ROM接続 CC000~ CDFFFF	ミュージック・ジェネレータ・ボードなし
						1					ミュージック・ジェネレータ・ボードあり
					0					拡張ROM接続 D0000~ D3FFFF	なし
					1						あり
	システム既定値 00h					0				拡張ROM接続 D4000~ D5FFFF	GPIBインターフェース・ボードなし
						1					GPIBインターフェース・ボードあり
						0				拡張ROM接続 CA000~ CBFFFF	なし
						1					あり
						0				拡張ROM接続 CE000~ CFFFFF	なし
						1					あり
										システム予約	



〈図4-2〉 アドレスの拡張



〈図4-3〉 拡張バスのスロットにより大きく異なる9本の信号線

端子番号	スロット #1の信号名	スロット #2, #3, #4の信号名と概要
A37	S00	INTA0 ..... I 8259A と外部 CPU のための信号
A38	S10	NOWAIT0 ... I メモリをノー・ウェイトで動かす要求信号
A39	S20	SALE1 ..... O AB171 ~ AB231 のラッチ要求
A40	LOCK0	MACS0 ..... I メモリ・ボードがアクセスされていることを示す信号, I/O 拡張ユニット用
B40	CPUKILL0	EXHRQ10 ... I 外部 CPU, DMA からのバス要求信号
B42	RQGT0	EXHLA10 ... O バス要求に対するアクノレッジ
B46	HLDA00	EXHLA20 ... O EXHLA10 に準ず
B47	HRQ00	EXHRQ20 ... I EXHRQ10 に準ず
B48	DMAHLD0	SBUSRQ1 ... O 内部 DMA からのバス要求信号

一が押すような構造になっています。また、ジャンパによりスロットを24ビットのモードか20ビットのモードかに固定することもできます。これで、バーの付いていない24ビット・デコードされた拡張基板(PC9801XA用)なども使用することができます。

20ビット・デコードの基板用のモードのしくみは多少複雑です。まず、VXには000000hから7FFFhまでの512KバイトにはRAMが本体に実装されていますから、このアドレスをデコードする拡張基板はスロットにささることはありません。つまりAB<sub>191</sub>が“0”のときにアクティブとなる基板はありません。これを利用して、AB<sub>201</sub>からAB<sub>231</sub>の4ビットがすべて0で、しかもAB<sub>191</sub>が“1”のときのみ、そのスロットのAB<sub>191</sub>を“1”とします(図4-2)。

つまり、プロテクト・モードでも080000hから0FFFFhの512Kバイトがアクセスされたときのみ、そのスロットは正しいアドレスが見え、その他の場合はAB<sub>191</sub>は“0”となって拡張基板のアクセスを禁止します。このため上位4ビットをデコードしなくても拡張ボード上でデコードしAB<sub>191</sub>をうまく使っているということです。

VXでは上記のアドレス信号の他に、スロットの9本の信号が今までのPC9801シリーズと異なっています。スロット#1はV30動作時のみ従来のPC9801シリーズと同じ信号が出ていて互換性を保っていますが、スロット#2, #3, #4は主に外部CPUとの接続を考えた変更が加えられています(図4-3)。本体内部の割り込みコントローラと外部のCPU間のため、外部CPUのバス要求、内部のDMAがリフレッシュを行うためのバス要求などの信号です。

### 4-3 80286のプロテクト・モード

80286のプロテクト・モードでは、リアル・モードよりも8086のオブジェクトをそのまま実行できる可能性

〈図4-4〉 VXのI/O空間とCPUの関係

I/O アドレス	R/W	内 容
F0h	W	80286 CPU のリセット
F2h	W	アドレス上位4ビットのマスク解除
F8h ↓ FFh		80287 が使用

〈図4-5〉  
VXのバンク・レジスタの  
I/Oアドレス

I/O アドレス		W/R	bit7	6	5	4	3	2	1	0
21h	DMAコントローラのバンク・レジスタ チャンネル 1	W	A23	A22	A21	A20	A19	A18	A17	A16
23h										
25h										
27h										
29h	バンク・レジスタ・モード	W						モード	チャンネル番号	

0	0	64KB境界
0	1	1MB境界
1	0	16MB境界

は少なくなります。

プロテクト・モードでは、1Gバイトの仮想メモリ空間と16Mバイトの実メモリ空間をVX上で使用できます。アドレス24ビット・フルデコードのRAMボードも市販されていますが、有効に利用できるのはVX用のBASICのみで、プロテクト・モードをサポートしたソフトウェアは現在のところありません。さしあたり、プロテクト・モードの空間にあるRAMをMS-DOSのRAMディスクとして使用する程度です。

リアル・モードからプロテクト・モードへは簡単に移行できますが、プロテクト・モードからリアル・モードへ戻ることは、CPUをリセットする以外に方法がありません。そのため、I/O空間のF0hへOUTすることにより、80286をリセットすることができます。このリセットは、CPUのみでVX本体をリセットするものではありませんから、リセット後のベクタをプロテクト・モード時に用意しておくことで、リアル・モードへの移行ができます。

その他に、プロテクト・モード用のアドレス上位4ビット・マスク解除や、80287(数値演算プロセッサ)用にI/O空間のF0hからF7hが使用されました(図4

4)。このI/O空間は予約されていたとはいえ、狭いPC9801シリーズのI/O空間の中の貴重な連続した16バイトだっただけに、アドレスのぶつかる基板も多いようです。

プロテクト・モードでは、16Mバイトのメモリ空間を扱うため、DMAコントローラも24ビット・アドレスへの対応がなされています。VMなどではDMACは16ビットのアドレスのみを発生し、バンク・レジスタが4ビットあり、合計20ビットで1Mバイトをアクセスしていました。DMAコントローラとバンク・レジスタは独立で、16ビットの64K境界をまたいでしまうような転送はできず、そのような場合は2度にわたる転送が必要でした。

VXではバンク・レジスタが8ビットとなり、24ビットのアドレスを発生させ、カウンタの機能もバンク・レジスタに付いたので、64K境界をまたいでしまう転送も可能になりました。したがって、互換性のため、バンク・レジスタは64Kバイト境界をもつモード、1Mバイト境界のモード、16Mバイト境界のモードと三つのモードをI/Oの29h番地により選択することができます(図4-5)。

## Cによる 科学技術計算

複素数の扱いからFFTのプログラミングまで

FORTRANやBASICで記述されていた科学技術計算のプログラムを、パーソナル・コンピュータ上のC言語に移植するために、それらのアルゴリズムを詳細に解析し、C言語による実現法を解説しました。ライブラリとしてすぐに役立つ数々のプログラム例をもとに、Cのテクニックが体得できます。

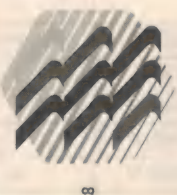
小池 慎一 著

A5判 340頁  
定価 2,300円  
送料 300円

絶賛発売中!!

Cによる  
科学技術計算

著者小池 慎一 小池 慎一 著



CQ出版社



# PC9801シリーズの拡張スロットの詳細

本章では、PC9801シリーズの拡張スロットの規格および拡張ボードの作り方について解説します。よく使用される信号線はだいたい決まっていますが、市販のボードを利用する時でもその中身を知っていると役に立ちます。

最近の16ビット・パソコンは、パソコンとはいいいがたいほどの処理能力をもっており、研究室や実際の現場での計測制御システムなどを構成するのに十分な力を発揮します。PC9801シリーズは、最近ではどの研究室にも1台はころがっているというのが現状のようです。

そういった中で、自分の目的に沿った拡張基板を製作し、仕事に役立てたいという要求はおのずと発生してくることでしょう。しかし、実際問題はスロット・バスの信号線の意味がよくわからないのでどうしたらよいかわからず、指をくわえるばかりといったユーザも少なくないに違いありません。

本章では、PC9801シリーズの拡張スロットに収納する拡張基板を設計する場合に必要な拡張バス・コネクタの定義について解説します。

PC9801シリーズの背面には表1-1に示すように機種による違いはありますが、最低2スロットから最大6スロットの拡張スロットが設けられています。スロットに収納する拡張基板の形状および寸法は、すべての機種において共通です。しかし、スロット・バスの信号線についてはクロック周波数の違いや、CPUの違いなどによって多少異なる部分があります。

また、PC9801以外ではスロット番号の一番大きなスロットと他のスロットでは、一部定義が異なります。しかしながら、これらの違いは例外的なものと考えられるため、実際の拡張基板ではすべての機種、スロットで動作するものがほとんどです。また、ユーザが新たに拡張基板を設計する場合も、すべての条件

〈表1-1〉機種とスロット・バスのタイプ

機種	スロットの個数	CPU	クロック (MHz)
PC9801	5	8086	5
PC9801E	6	8086	5/8
PC9801F	4	8086	5/8
PC9801M	4	8086	5/8
PC9801U	2	V30	8
PC9801VF	4	V30	8
PC9801VM	4	V30	8/10

で動作するように設計すべきでしょう。

本体内に収納可能な拡張基板数は、最大のPC9801Eでも6枚までですが、I/O拡張ユニットPC9811Kを利用することによって、さらに5スロットの増設が可能です。

## 1 拡張基板の外形と信号線の意味

### 1-1 拡張基板の形状および寸法

図1-1に拡張基板の形状および寸法を示します。

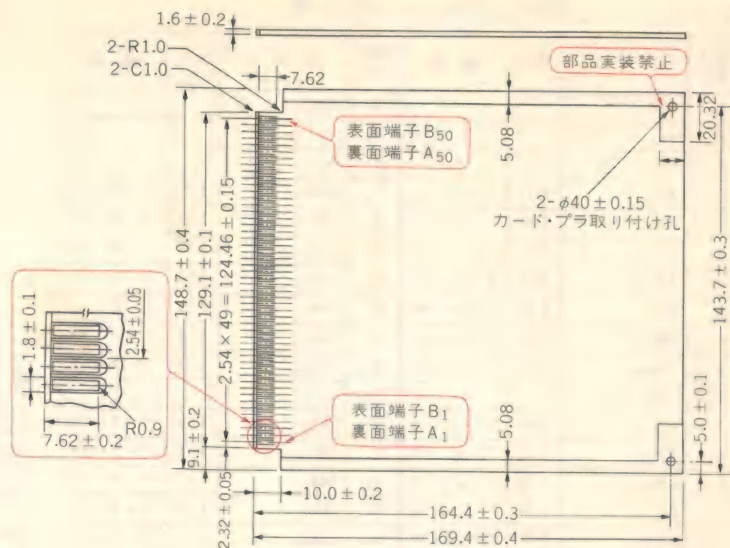
図の左側がコンピュータ本体に挿入される側で、基板の左端には本体内のバス・コネクタに接続するためのカード・エッジ・パターンがプリントされています。カード・エッジ・パターンには金メッキを施すのが普通ですが、このときにパターンを必要な信号線だけにすると使用する金の量が少なくてすみ、基板のコスト・ダウンになるかもしれません。

基板の右端の両側には基板をスロットから容易に引き抜くために、カード・ブラが取り付けられます。基板をスロットに挿入するのは比較的容易なのですが、引き抜くのはカード・ブラなしでは大変な労力を要します。

外部との信号のやり取りが必要な場合には、基板の左端にコネクタが取り付けられます。しかし、これによって従来のスロットのフタは取り付けられなくなります。純正の拡張基板では、専用のフタが基板と一体になって付属していますが、そのかわりカード・ブラが付いていないのが普通です。着脱の容易さを考慮すると、フタよりもカード・ブラを優先したほうがよいかもしれません。

拡張基板の高さに対する規定は特にありませんが、スロットの高さ方向のピッチが25mmですから、実際問題として基板上面から約20mmが限界です。余裕をもって18mm程度に抑えるのが無難でしょう。この数値は通常の実装方法では特に問題ありませんが、基板を2階構造にしてコネクタなどで接続して使用する場合には注意が必要です。2階構造にした場合、2枚の

〈図1-1〉 拡張基板の形と寸法



基板の間隔は10mm程度になり、実際に実装可能な部品の高さは1階2階ともに7～8mm程度でしょう。この場合、ICソケットを利用したり背の高い部品(コンデンサなど)を実装しようとする、思わぬ誤算を招く結果になりかねません。

## 1-2 バス・コネクタの形状と信号線

バス・コネクタは、図1-1でもわかるように、基板の裏表を利用したカード・エッジ・コネクタです。両面ともに2.54mmピッチの50ピンで、合計で100ピンになります。図1-1を基板の表側、すなわち部品を実装する面から見た図と考えると、図の下側のピンから上に向かって順に1から50までの端子番号が付けられています。また、裏側はA、表側はBとして裏表を区別します。

表1-2にバス・コネクタの信号線の一覧を示します。注に示したように一部の信号線の定義が機種やクロック周波数およびスロット番号によって異なっています。

## 1-3 各信号線の意味

次に各信号線の意味を解説します。ここで、**信号名の最後の数字はアクティブとなる論理を示すもので**，“0”の場合はアクティブ“L”（負論理），“1”の場合はアクティブ“H”（正論理）を意味します。また、特に断りがない限りアクティブになった場合の意味を表すものとし、ユーザが拡張基板を設計する場合に使用の可能性が高いものに対しては比較的詳しく行い、それ以外のものに対しては簡単な説明にとどめています。

### ▶ AB<sub>001</sub>～AB<sub>191</sub>：アドレス・バス

通常は出力モードで、CPUのAD<sub>0</sub>～AD<sub>15</sub>およびS<sub>3</sub>～S<sub>6</sub>からセパレートされた20ビットのアドレスが出

力されます。本体上のCPUがディセーブルされたり、外部のDMACがセレクトされると入力モードになります。コネクタには、AB<sub>201</sub>～AB<sub>231</sub>のさらに4ビットのアドレス・ラインが定義されています。これらは新しいPC9801VX/XLシリーズでは使用されていますが、従来のPC9801シリーズでは未使用です（詳細は第1章を参照）。

### ▶ BHE<sub>0</sub>：バス・ハイ・イネーブル

データ・バスの上位バイト（奇数アドレス）に対するアクセスを行うことを示します。下位バイト（偶数アドレス）に対してはAB<sub>001</sub>が同様な働きをします。すなわち、AB<sub>001</sub>とBHE<sub>0</sub>の組み合わせによって、表1-3のような3種類のアクセス・パターンが存在します。

ここでの**ワード・アクセスとは、偶数アドレスから始まる2バイトのアクセスのことです**。奇数アドレスから始まるワード・アクセスの場合は、CPUが2回のバイト・アクセスに置き換えます。

### ▶ DB<sub>001</sub>～DB<sub>151</sub>：データ・バス

16ビットの双方向データ・バスです。DB<sub>001</sub>～DB<sub>071</sub>は、下位バイトすなわち偶数アドレスのデータのやり取りに用いられ、DB<sub>081</sub>～DB<sub>151</sub>は上位バイトすなわち奇数アドレスのデータのやり取りに用いられます。

### ▶ IOR<sub>0</sub>：I/Oリード

I/Oデバイスからの読み込みを示すストローブ信号です。CPUがIN命令を実行したときにアクティブになります。

### ▶ IOW<sub>0</sub>：I/Oライト

I/Oデバイスへの書き込みを示すストローブ信号です。CPUがOUT命令を実行したときにアクティブになります。

### ▶ MRC<sub>0</sub>：メモリ・リード

メモリからの読み込みを示すストローブ信号です。



〈表1-2〉 スロット・バス端子一覧

端子番号	信号名	方向	機能	端子番号	信号名	方向	機能
A <sub>1</sub>	GND			B <sub>1</sub>	GND		
A <sub>2</sub>	V <sub>1</sub>			B <sub>2</sub>	V <sub>1</sub>		
A <sub>3</sub>	V <sub>2</sub>			B <sub>3</sub>	V <sub>2</sub>		
A <sub>4</sub>	AB <sub>001</sub>	I/O	アドレス・バス	B <sub>4</sub>	DB <sub>001</sub>	I/O	データ・バス
A <sub>5</sub>	AB <sub>011</sub>	I/O	アドレス・バス	B <sub>5</sub>	DB <sub>011</sub>	I/O	データ・バス
A <sub>6</sub>	AB <sub>021</sub>	I/O	アドレス・バス	B <sub>6</sub>	DB <sub>021</sub>	I/O	データ・バス
A <sub>7</sub>	AB <sub>031</sub>	I/O	アドレス・バス	B <sub>7</sub>	DB <sub>031</sub>	I/O	データ・バス
A <sub>8</sub>	AB <sub>041</sub>	I/O	アドレス・バス	B <sub>8</sub>	DB <sub>041</sub>	I/O	データ・バス
A <sub>9</sub>	AB <sub>051</sub>	I/O	アドレス・バス	B <sub>9</sub>	DB <sub>051</sub>	I/O	データ・バス
A <sub>10</sub>	AB <sub>061</sub>	I/O	アドレス・バス	B <sub>10</sub>	DB <sub>061</sub>	I/O	データ・バス
A <sub>11</sub>	GND			B <sub>11</sub>	GND		
A <sub>12</sub>	AB <sub>071</sub>	I/O	アドレス・バス	B <sub>12</sub>	DB <sub>071</sub>	I/O	データ・バス
A <sub>13</sub>	AB <sub>081</sub>	I/O	アドレス・バス	B <sub>13</sub>	DB <sub>081</sub>	I/O	データ・バス
A <sub>14</sub>	AB <sub>091</sub>	I/O	アドレス・バス	B <sub>14</sub>	DB <sub>091</sub>	I/O	データ・バス
A <sub>15</sub>	AB <sub>101</sub>	I/O	アドレス・バス	B <sub>15</sub>	DB <sub>101</sub>	I/O	データ・バス
A <sub>16</sub>	AB <sub>111</sub>	I/O	アドレス・バス	B <sub>16</sub>	DB <sub>111</sub>	I/O	データ・バス
A <sub>17</sub>	AB <sub>121</sub>	I/O	アドレス・バス	B <sub>17</sub>	DB <sub>121</sub>	I/O	データ・バス
A <sub>18</sub>	AB <sub>131</sub>	I/O	アドレス・バス	B <sub>18</sub>	DB <sub>131</sub>	I/O	データ・バス
A <sub>19</sub>	AB <sub>141</sub>	I/O	アドレス・バス	B <sub>19</sub>	DB <sub>141</sub>	I/O	データ・バス
A <sub>20</sub>	AB <sub>151</sub>	I/O	アドレス・バス	B <sub>20</sub>	DB <sub>151</sub>	I/O	データ・バス
A <sub>21</sub>	GND			B <sub>21</sub>	GND		
A <sub>22</sub>	AB <sub>161</sub>	I/O	アドレス・バス	B <sub>22</sub>	+12V		
A <sub>23</sub>	AB <sub>171</sub>	I/O	アドレス・バス	B <sub>23</sub>	+12V		
A <sub>24</sub>	AB <sub>181</sub>	I/O	アドレス・バス	B <sub>24</sub>	IR <sub>31</sub>	I	INT <sub>0</sub>
A <sub>25</sub>	AB <sub>191</sub>	I/O	アドレス・バス	B <sub>25</sub>	IR <sub>51</sub>	I	INT <sub>1</sub>
A <sub>26</sub>	AB <sub>201</sub>	I/O	アドレス・バス	B <sub>26</sub>	IR <sub>61</sub>	I	INT <sub>2</sub>
A <sub>27</sub>	AB <sub>211</sub>	I/O	アドレス・バス	B <sub>27</sub>	IR <sub>91</sub>	I	INT <sub>3</sub> (5 <sup>HD</sup> )
A <sub>28</sub>	AB <sub>221</sub>	I/O	アドレス・バス	B <sub>28</sub>	IR <sub>101</sub> /IR <sub>111</sub>	I	INT <sub>4</sub> /INT <sub>42</sub> <sup>(*)1</sup>
A <sub>29</sub>	AB <sub>231</sub>	I/O	アドレス・バス	B <sub>29</sub>	IR <sub>121</sub>	I	INT <sub>5</sub>
A <sub>30</sub>	INT <sub>0</sub>	O		B <sub>30</sub>	IR <sub>131</sub>	I	INT <sub>6</sub>
A <sub>31</sub>	GND			B <sub>31</sub>	GND		
A <sub>32</sub>	IOCHK <sub>0</sub>	I	外部NMI <sup>(*)1</sup>	B <sub>32</sub>	-12V		
A <sub>33</sub>	IOR <sub>0</sub>	I/O	コマンド	B <sub>33</sub>	-12V		
A <sub>34</sub>	IOW <sub>0</sub>	I/O	コマンド	B <sub>34</sub>	RESET <sub>0</sub>	O	RESET
A <sub>35</sub>	MRC <sub>0</sub>	I/O	コマンド	B <sub>35</sub>	DACK <sub>00</sub>	O	5 <sup>HD</sup>
A <sub>36</sub>	MWC <sub>0</sub>	I/O	コマンド	B <sub>36</sub>	DACK <sub>30</sub> /DACK <sub>20</sub>	O	AUX
A <sub>37</sub>	S <sub>00</sub>	I/O	S <sub>0</sub>	B <sub>37</sub>	DRQ <sub>00</sub>	I	5 <sup>HD</sup>
A <sub>38</sub>	S <sub>10</sub>	I/O	S <sub>1</sub>	B <sub>38</sub>	DRQ <sub>20</sub> /DRQ <sub>20</sub>	I	AUX
A <sub>39</sub>	S <sub>20</sub>	I/O	S <sub>2</sub>	B <sub>39</sub>	WORD <sub>0</sub>	I	
A <sub>40</sub>	LOCK <sub>0</sub>	I/O		B <sub>40</sub>	CPKILL <sub>0</sub>	I	
A <sub>41</sub>	GND			B <sub>41</sub>	GND		
A <sub>42</sub>	CPUENB <sub>10</sub>	O		B <sub>42</sub>	RQGT <sub>0</sub>	I	バスの解放要求
A <sub>43</sub>	RFSH <sub>0</sub>	O		B <sub>43</sub>	DMATC <sub>0</sub>	O	END OF PROCESS
A <sub>44</sub>	BHE <sub>0</sub>	I/O		B <sub>44</sub>	NMI <sub>0</sub>	O	
A <sub>45</sub>	IORDY <sub>1</sub>	I		B <sub>45</sub>	MWE <sub>0</sub>	O	
A <sub>46</sub>	SCLK <sub>1</sub>	O	7.9872MHz <sup>(*)2</sup>	B <sub>46</sub>	HLDA <sub>00</sub>	O	
A <sub>47</sub>	S18CLK <sub>1</sub>	O	307.2kHz	B <sub>47</sub>	HRQ <sub>00</sub>	I	
A <sub>48</sub>	POWER <sub>0</sub>	O	電源確認信号	B <sub>48</sub>	DMAHLD <sub>0</sub>	I	
A <sub>49</sub>	+5V			B <sub>49</sub>	+5V		
A <sub>50</sub>	+5V			B <sub>50</sub>	+5V		

〈表1-3〉 データ・バスのアクセス・パターン

BHE <sub>0</sub>	AB <sub>001</sub>	意味
0	0	ワード・アクセス
0	1	上位バイト(奇数アドレス)
1	0	下位バイト(偶数アドレス)
1	1	

〈表1-4〉 割り込み要求信号線の割り当て

信号名	接続装置	ベクタ番号	備考
IR <sub>31</sub>	未割り当て	OBh	INT <sub>0</sub>
IR <sub>51</sub>	カセット磁気テープ	ODh	INT <sub>1</sub>
IR <sub>61</sub>	未割り当て	OEh	INT <sub>2</sub>
IR <sub>91</sub>	5インチ固定ディスク	11h	INT <sub>3</sub>
IR <sub>101</sub>	640Kバイト・フロッピー・ディスク	12h	INT <sub>41</sub>
IR <sub>111</sub>	1Mバイト・フロッピー・ディスク	13h	INT <sub>42</sub>
IR <sub>121</sub>	未割り当て	14h	INT <sub>5</sub>
IR <sub>131</sub>	マウス	15h	INT <sub>6</sub>

〈表1-5〉 NMIの制御

I/Oアドレス	機能(書き込み時)
50h	NMIを禁止する
52h	NMIを許可する

〈表1-6〉 PC9801シリーズのクロック周波数

9.8304MHz	10Mモード
7.9872MHz	8Mモード
4.9152MHz	5MモードおよびPC9801

〈表1-7〉 CPUのステータス(バス・サイクル)の意味

S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	意味
0	0	0	インタラプト・アクノレッジ
0	0	1	リードI/Oポート
0	1	0	ライトI/Oポート
0	1	1	ホールド
1	0	0	コード・アクセス
1	0	1	リード・メモリ
1	1	0	ライト・メモリ
1	1	1	受動

- (\*)1 スロット番号が一番大きいスロットの場合は後者、その他のスロットは前者。  
 (\*)2 CPUクロックにより、4.9152MHzまたは9.8304MHzになることもある。  
 (\*)3 U2では未使用。

▶ **MWC<sub>0</sub>** : メモリ・ライト

メモリへの書き込みを示すストローブ信号です。

▶ **MWE<sub>0</sub>** : メモリ・ライト・イネーブル

MWC<sub>0</sub>よりも遅れたタイミングのメモリへの書き込みストローブ信号です。主として拡張メモリに対するDRAMの書き込みタイミング信号として用いられます。

▶ **RFSH<sub>0</sub>** : リフレッシュ

バスがDRAMのリフレッシュのために、占有されていることを示します。

▶ **IR<sub>31</sub> ~ IR<sub>131</sub>** : 割り込み要求信号

外部からCPUに対してマスカブル割り込みをかけるための入力信号です。各信号のポジティブ・エッジ(“L” から “H” へ変化する瞬間)に割り込みがかかります。拡張スロットから入力可能な割り込みチャンネルは合計8チャンネルですが、実際に割り当てられている端子は7本です。

すなわち、IR<sub>101</sub>とIR<sub>111</sub>とは同じ端子(B<sub>28</sub>)に割り当てられており、スロット番号が一番大きなスロット(例えばスロットが全部で4スロットあるPC9801VMなどではスロット#4)ではIR<sub>111</sub>、他のスロットではIR<sub>101</sub>になります。ただし、PC9801の場合にはすべてのスロットがIR<sub>101</sub>に割り当てられています。

これらの割り込み要求信号線は表1-4のように予約されており、ユーザが割り込みを使用する場合には、未割り当ての割り込みを使用しなければなりません。

▶ **IOCHK<sub>0</sub>** : NMI要求信号

CPUに対してノンマスカブル割り込みをかけるための入力信号です。本信号のネガティブ・エッジにより割り込みがかかります。

ノンマスカブル・インタラプト(NMI)は、ソフトウェアからのマスク(禁止)制御ができない割り込みで、通常はメモリ・パリティ・エラー検出時に使用しています。ただし、NMIコントロール・ポートに書き込みを行うことにより、ハードウェアで禁止することが可能です(表1-5)。

▶ **INT<sub>0</sub>** : インタラプト

各割り込み要求に対して割り込みコントローラ(8259A)が応答したことを示します。

▶ **NMI<sub>0</sub>** : ノンマスカブル・インタラプト

ノンマスカブル割り込みがあったことを示します。

▶ **SCLK<sub>1</sub>** : システム・クロック

CPUのクロックです。機種およびクロック切り替えスイッチによって、表1-6に示したような周波数のクロックが出力されます。また、CPUの違いによってクロックのデューティ比が異なります。詳しくはタイミングの項を参照してください。

▶ **S<sub>18</sub>CLK<sub>1</sub>** : 307.2kHz

307.2kHzのクロック信号です。シリアル通信回線

用のボーレート・クロックとして用いると便利です。

▶ **POWER<sub>0</sub>** : 電源確認信号

▶ **RESET<sub>0</sub>** : リセット信号

リセットは、+DC 5 Vが4.75V以下になるか、本体のリセット・スイッチが押されることによってアクティブになります。

▶ **DRQ<sub>00</sub> ~ DRQ<sub>30</sub>** : DMA要求信号

▶ **DACK<sub>00</sub> ~ DACK<sub>30</sub>** : DMAアクノレッジ信号

ダイレクト・メモリ・アクセス(DMA)を用いてデータ転送を行うときのハンドシェイク信号線です。

DRQ<sub>00</sub>とDACK<sub>00</sub>は、5インチ固定ディスクのために用意されています。

DRQ<sub>20</sub>とDACK<sub>20</sub>は、1 Mバイト・フロッピー・ディスクのために用意されており、前述のIR<sub>111</sub>と同じようにスロット番号が一番大きなスロットにのみ割り当てられています。一方、他のスロットでは同じ端子がDRQ<sub>30</sub>とDACK<sub>30</sub>に割り当てられており、これらは640Kバイト・フロッピー・ディスクのために用意されています。

▶ **WORD<sub>0</sub>** : ワード/バイト

DMA転送時にワード転送をする場合にアクティブにします。旧タイプのシリーズでは使用されていましたが、最近の機種では未使用です。

▶ **DMATC<sub>0</sub>** : DMAターミナル・カウント

DMA転送時の最終ワード(またはバイト)のときにアクティブになります。

▶ **DMAHLD<sub>0</sub>** : DMAホールド

内部のDMAの要求をすべてインアクティブにし、内部DMAが動作しないようにする信号線です。主に外部DMAがバスを占領するために使用します。

DMAHLD<sub>0</sub>はDRAMのリフレッシュを止めますので、長時間(約140クロック以上)アクティブにしてはいけません。

▶ **HRQ<sub>00</sub>** : ホールド・リクエスト信号

CPUにホールドを要求する信号です。CPUはホールド要求されるとウェイトの状態になります。CPUを確実にホールドするためには内部のS<sub>4</sub>からS<sub>0</sub>までHRQ<sub>00</sub>をアクティブにする必要があります。

▶ **HLDA<sub>00</sub>** : ホールド・アクノレッジ信号

CPUがホールド状態になったことを示す信号線です。

▶ **CPUENB<sub>10</sub>** : CPUイネーブル信号

CPUがバスを使用しているときにアクティブになる信号線です。

▶ **IORDY<sub>1</sub>** : I/Oレディ

CPUをウェイト状態にするための信号線です。CPUのスピードに対して、アクセス速度が遅いメモリやI/Oデバイスを使用する場合に用います。IORDY<sub>1</sub>が“L”レベルのとき、CPUはウェイト・サ



イクルを繰り返し、“H”レベルになって初めて次のサイクルに進みます。

CPUがI/Oをアクセスする場合には、内部で自動的にウェイトが入ります(5MHzモード/1ウェイト, 8MHzモード/2ウェイト, 10MHzモード/3ウェイト)。

なお、 $IORDY_1$ はオープン・コレクタで出力する必要があります。

▶  $S_{00}$ ,  $S_{10}$ ,  $S_{20}$ : CPUステータス信号

CPUのマキシマム・モードにおける $S_0$ ,  $S_1$ ,  $S_2$ のステータス信号がそのまま出力されています。 $S_0$ ,  $S_1$ ,  $S_2$ の組み合わせによって、表1-7のようなステータスを意味します。

▶  $RQ/GT_0$ : リクエスト/グラント信号

CPUのマキシマム・モードにおける $RQ/GT$ 信号で

す。

▶  $LOCK_0$ : ロック信号

CPUのマキシマム・モードにおける $LOCK$ 信号です。

▶  $CPKILL_0$

CPUのアドレス/データ・バス・バッファをディセーブルにしてCPUをバスから切り放すための信号です。

▶  $GND$ : グラウンド

▶  $+5V$ :  $+5V$ 電源ライン

▶  $+12V$ :  $+12V$ 電源ライン

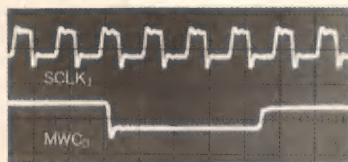
▶  $-12V$ :  $-12V$ 電源ライン

▶  $V_1$ ,  $V_2$ : オプション電源ライン

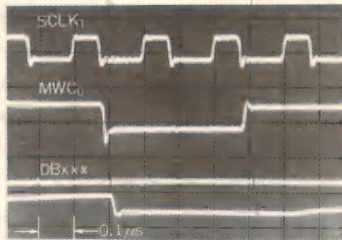
オプション電源ラインには本体から電源は供給されていません。

## 拡張バスの各種信号の実例

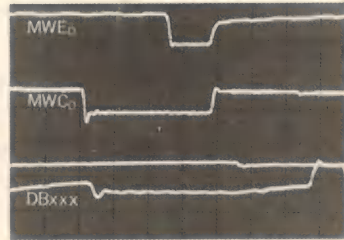
写真Aは、(a)が8MHzのPC9801Fのシステム・クロックとライトのストロープで、下段が5MHzのときのクロックとストロープです。5MHz時にはウェイト・サイクルは入っていませんが、8MHz時には1ウェイト・サイクルが入っていますので、**どちらもストロープの長さは400ns以上あります。**(b)のライトのストロープとデータから $MWC_0$ が下がった時点では、まだデータは確定していないことがわかります。



(a) 8 MHzクロック

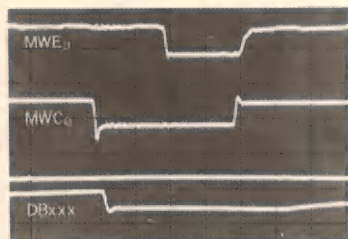


(b) 5 MHzクロック

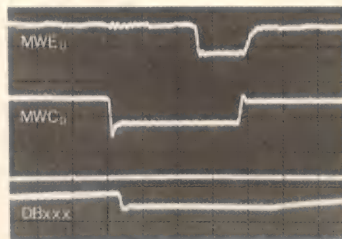


(a) 8 MHz

〈写真A〉 PC9801Fのシステム・クロックとメモリ・ライト・ストロープ



(a) 5 MHz

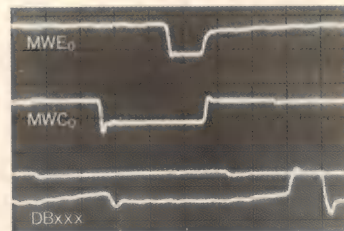


(b) 8 MHz

〈写真B〉 PC9801Fのデータのリード/ライト波形

ライト時のデータが確定してからのストロープが欲しい場合は、 $MWE_0$ を用います。写真Bの(a)は5MHz時の $MWE_0$ ,  $MWC_0$ , およびデータ、(b)は8MHz時のものです。PC9801VMの場合には、写真Cのようになります。**8MHz時も10MHz時も1ウェイトが入っています。10MHz時のストロープの幅は約300nsと短くなっています(写真D)。**

写真EはI/Oのライトのタイミングです。 $IORDY_1$ を使用しなくても、ある程度のウェイト・サイクルが入り、**360ns程度のストロープ幅**があります。データ・バスは十分に確定しています。



(b) 10 MHz

〈写真C〉 PC9801VMのデータのリード/ライト波形

## 2 拡張スロットの電氣的仕様

### 2-1 各信号線のドライブ能力

スロット・バスの各信号線は、例外的なものを除いてすべてのスロットに対して並列に配線されています。本体内の信号線のドライブ能力は有限(ほとんどは74LS245相当の3ステート・バッファで出力されている)ですから、1スロット当たりの負荷容量を規定しておかないと、複数のスロットに同時に拡張基板を装着した場合に、本体のドライブ能力を超えてしまいます。この場合、CPUが誤動作を起こすだけでなく、最悪の場合には本体のICを壊す結果になります。

また、スロットに装着された拡張基板の出力(デー

タ出力など)信号線は、本体内の回路をドライブするとともに、他のスロットの拡張基板をもドライブする必要があります。したがって、最大負荷の場合を考えて十分なドライブ能力をもっていることが必要です。

表2-1に各信号線の1スロット当たりの最大入力電流(許容負荷容量)と、拡張基板に要求される最小出力電流(ドライブ能力)の規定値を示します。前者が空欄になっているのは、入力専用の信号線です。また、後者が“不可”となっているのは出力専用の信号線です。

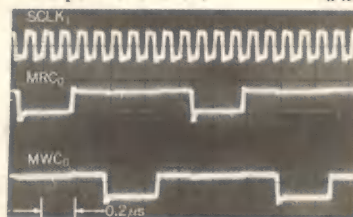
アドレス/データ・バスや一般的なコントロール信号線では、1スロット当たりに許されている $I_{IL}$ (ファンイン)は $-0.8\text{mA}$ です。一般のLS TTLの $I_{IL}$ は $-0.4\text{mA}$ ですから、1枚の拡張基板において二つの入力まで並列に接続できます。ただし、TTLによっては入力が通常の2倍必要なものもありますので注意が必要

写真Eを見るとPC9801FとVMでは、同じ8MHzのクロックでもデューティ・サイクルが違うのがよくわかります。

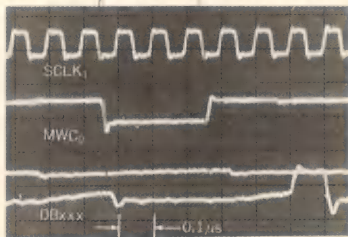
次にリフレッシュ信号を見てみます(写真F)。VMのリフレッシュは約 $16\mu\text{s}$ に一度行われており、256ロウ・アドレスを4msで読み出していますので、タイミングとしては十分です。FのRFSH<sub>0</sub>は $28\mu\text{s}$ に一度です。これでは256ロウ・アドレスに7msもかかってしまいます。

写真Gは、8086のストリング命令MOVESWを実行しているときの、10MHz時のVMのリードとライ

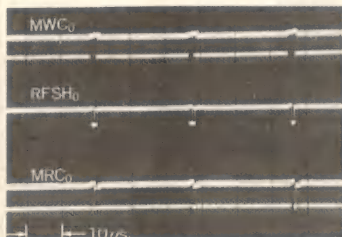
トのストロブです。見てわかるとおり、わずか10サイクルで1ワードの転送を行いますので、転送能力は2Mbpsになります。〈秋葉澄伸〉



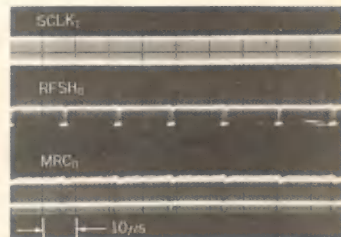
〈写真G〉 ストリング命令実行時のリード/ライト・ストロブ



〈写真D〉 PC9801VM(10MHz)のシステム・クロックとメモリ・ライト・ストロブ

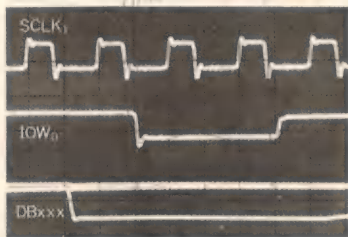


(a)PC9801F(8MHz)

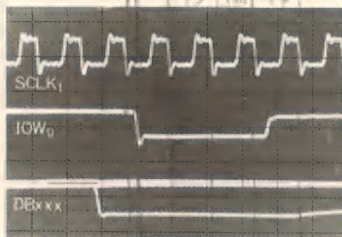


(b)PC9801VM(10MHz)

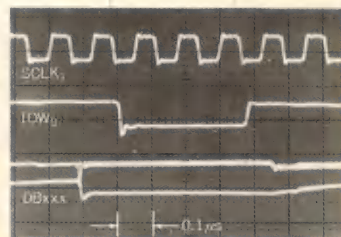
〈写真F〉 リフレッシュ信号



(a)PC9801F(5MHz)



(b)PC9801F(8MHz)



(c)PC9801VM(8MHz)

〈写真E〉 PC9801のI/Oライト・タイミング



です(アドレス・デコードなどに頻繁に用いられるLS266はその代表的な例)。

一方、これらの信号線を拡張基板がドライブする場合には、 $I_{OL}$ (ファンアウト)が最低12mA要求されています。この値は一般のLS TTLでは満たすことは不可能で、バッファ・タイプのLS TTL(74LS245など)を使用しなければなりません。もちろん、8255AなどのペリフェラルLSIでも直接ドライブすることはできません。ただし、 $I_{OL}$ が8 mAでよい信号線は一般のLS TTLでドライブ可能です。

IOCHK<sub>0</sub>やIORDY<sub>1</sub>などの信号線は、バス上でアンド・タイ(ワイヤードOR)にされる可能性があります。したがって、これらの信号線はオープン・コレクタ(LS03などを用いる)で出力する必要があります。

2-2 電源容量

スロット・バスには、電源として+5 V、+12V、-12Vの3種類が供給されています。それぞれの電源の1スロットあたりに許されている容量は、表2-2の

〈表2-2〉 1スロット当たりの電源容量

電源	変動率	PC9801/E/F/M	PC9801U/VF/VM
+5 V	±5%以内	0.5 A	0.5 A
+12V	±10%以内	0.06 A	0.05 A
-12V	±10%以内	0.07 A	0.07 A

ように規定されています。また、全スロット合計の最大容量は1スロット当たりの容量にスロットの個数を掛けた値です。この容量値を超える拡張基板を作成した場合には、ボードの組み合わせによって全スロット合計の許容値を超えてしまい、電源を破壊することもあります。

2-3 信号線のタイミング

▶ システム・クロック

システム・クロック(SCLK<sub>1</sub>)の周波数は、機種やクロック切り替えスイッチの位置によって異なります。また、同じクロック周波数でも、CPUが8086の場合とV30との場合ではクロックのデューティ比が異なっ

〈表2-1〉 各信号線のドライブ能力

信号名	1 スロット当たりの最大入力電流		外部ロジックがドライブする時の最小出力電流	
	$I_{IL}(\text{mA})$	$I_{IH}(\mu\text{A})$	$I_{OL}(\text{mA})$	$I_{OH}(\text{mA})$
AB <sub>001</sub> ～AB <sub>191</sub>	-0.8	40	12	-1.2
BHE <sub>0</sub>				
DB <sub>001</sub> ～DB <sub>151</sub>				
IOR <sub>0</sub>				
IOW <sub>0</sub>				
MRC <sub>0</sub>				
NWC <sub>0</sub>				
MWF <sub>0</sub>				
RFSH <sub>0</sub>	全スロットで-0.8	全スロットで40	12	-1.2
IR <sub>31</sub> ～IR <sub>131</sub>	—	—	8	-0.4
IOCHK <sub>0</sub>	—	—		
INT <sub>0</sub>	-0.8	40	不可	
NMI <sub>0</sub>				
SCLK <sub>1</sub>				
S <sub>18</sub> CLK <sub>1</sub>				
POWER <sub>0</sub>				
RESET <sub>0</sub>				
DRQ <sub>00</sub> ～DRQ <sub>30</sub>	—	—	8	-0.4
DACK <sub>00</sub> , DACK <sub>30</sub>	全スロットで-1.6	全スロットで80	不可	
DMATC <sub>0</sub>	全スロットで-1.6	全スロットで80	8	-0.4
DMAHLD <sub>0</sub>	—	—		
HRQ <sub>00</sub>	—	—		
HRDA <sub>00</sub>	-0.8	40	不可	
CPUENB <sub>10</sub>				
IORDY <sub>0</sub>	—	—	8	-0.4
S <sub>00</sub> , S <sub>10</sub> , S <sub>20</sub>	全スロットで-0.4	全スロットで20		
RQ/GT <sub>0</sub>	—	—		
LOCK <sub>0</sub>	全スロットで-0.4	全スロットで20		
CPKILL <sub>0</sub>	—	—		

ています。

すなわち、8086の場合には“L”と“H”のデューティ比が2対1だったのに対して、V30の場合にはその比が1対1になっています(図2-1および表2-3参照)。したがって、システム・クロックを用いて各種タイミングを作成する場合には、**どちらのCPUでも動作可能なように注意して設計する必要があります。**

#### ▶メモリ・リード・サイクル

メモリ・リード・サイクルは、 $T_2$ ステートの始まりで $MRC_0$ がアクティブになることからスタートします。 $T_2$ ステートの終わりに $IORDY_1$ が“L”であれば、CPUはウェイト・サイクル( $T_w$ ステート)に入ります。その後、 $IORDY_1$ が“H”になると $T_w$ ステートから $T_4$ ステートに移行し、その瞬間にデータがCPUに読み込まれます。システム・クロックが5MHzのときはウェイト・サイクルは入りませんが、8MHzまたは10MHzのときには**1クロックのウェイト・サイクルが自動的に挿入されます。**ただし、10MHzモードで、オプションROM空間をアクセスする場合には、2クロックのウェイトが入ります(図2-2および表2-4参照)。

メモリ素子に要求されるアクセス・タイム $t_{acc}$ は、バス・パッファなどのディレイ・タイムを無視すると、次のようにして求めることができます。

$$t_{acc} = (2+n) \times t_{cy} - t_{CLMRL} - t_{DVCL} \dots\dots\dots(1)$$

ここで、 $n$ は挿入されるウェイト・サイクル数です。一番厳しいと考えられる10MHzの場合について試算してみると、

$$3 \times 101.73 - 38 - 28 \approx 239(\text{ns})$$

という結果になります。

#### ▶I/Oリード・サイクル

I/Oリード・サイクルでは、CPUが8086の場合とV30の場合とで $IOR_0$ がアクティブになるタイミングに違いがあります。8086では、メモリ・リード・サイ

クルと同様に、 $T_2$ ステートの始めにアクティブになりますが、V30の場合には $T_3$ ステートの始めになって初めてアクティブになります(図2-2および表2-4参照)。

したがって、8086の場合のI/Oに要求されるアクセス・タイムは、(1)式と同様にして(もちろん $t_{CLMRL}$ は $t_{CLIRL}$ に変更する)求められるのですが、V30の場合には(1)式の2を1に変えて計算しなければなりません。すなわちV30の場合には、より厳しいアクセス・タイムが要求されるのです。

一般に、I/Oデバイスはメモリに対してアクセス・スピードが遅いものも多く、クロックが5MHz、8MHz、10MHzの場合にそれぞれ1, 2, 3サイクルのウェイト・サイクルが自動的に挿入されています。

アクセス・スピードが最も要求されるV30の8MHzモード(10MHzの場合にはウェイトのサイクル数が多いため、8MHzより楽になる)の場合のアクセス・タイムを計算してみると、

$$3 \times 125.2 - 50 - 38 \approx 287(\text{ns})$$

となります。

#### ▶メモリ・ライト・サイクル

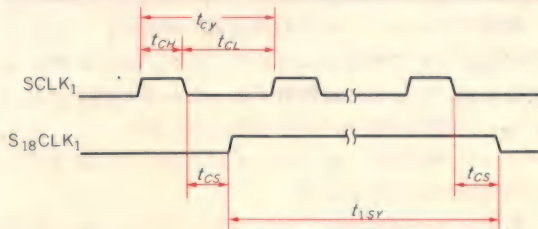
メモリ・ライト・サイクルは、メモリ・リード・サイクルと同様に $T_2$ の始めに $MWC_0$ がアクティブになることからスタートします。ただし、このときに**データ・バスの内容が確定している保証はありません。**データ・バスが確定後のストロブ信号として、 $MWE_0$ が $T_3$ ステート( $T_w$ ステートが入った場合には最後の $T_w$ ステート)の始めにアクティブになります。

$T_3$ ステート( $T_w$ ステートが入った場合には $T_w$ ステート)が終わると $MWC_0$ および $MWE_0$ がインアクティブになり、その後データ・バスのデータが不定になります。すなわち、 **$MWE_0$ がアクティブの間は、データ・バスのデータは確定していること**になります。

しかし、 $MWE_0$ がアクティブになっているのが保証されるのは、1クロック・サイクル未満の短期間ですし、 $MWE_0$ のポジティブ・エッジからのデータのホールド・タイムが $MWC_0$ の場合に比べて短いので、使用するには注意が必要です(図2-3および表2-5参照)。

メモリ・ライト・サイクルにおいても自動的にウェイト・サイクルが挿入され、そのサイクル数はメモリ・リード・サイクルと同様です。

〈図2-1〉システム・クロック

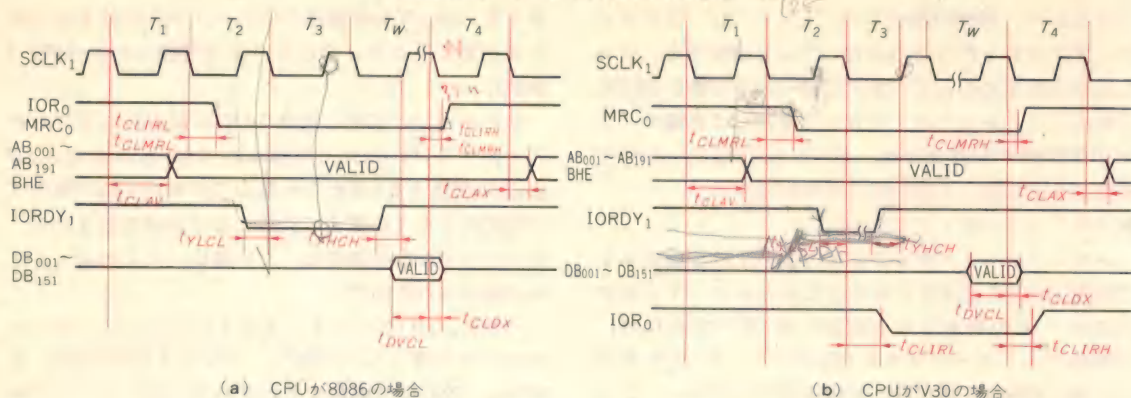


〈表2-3〉システム・クロックのタイミング

記号	パラメータ	5MHz mode(8086)		8MHz mode(8086)		8MHz mode(V30)		10MHz mode(V30)	
$t_{cy}$	SCLK Cycletime	203.45		125.20		125.20		101.73	
記号	パラメータ	min (ns)	max (ns)	min (ns)	max (ns)	min (ns)	max (ns)	min (ns)	max (ns)
$t_{CH}$	SCLK hightime	70	84	43	57	46	79	38	64
$t_{CL}$	SCLK low time	120	134	68	82	56	69	46	56
$t_{18y}$	S <sub>18</sub> CLK cycletime	307.200(Hz)		307.200(Hz)		307.200(Hz)		307.200(Hz)	
$t_{CS}$	S <sub>18</sub> CLK Delytime	19	67	83	133	—	規定しない	—	



〈図2-2〉 リード・サイクルのタイムチャート



(a) CPUが8086の場合

(b) CPUがV30の場合

〈表2-4〉  
リード・サイクルの  
タイミング

記号	パラメータ	5MHz mode(8086)		8MHz mode(8086)		8MHz mode(V30)		10MHz mode(V30)	
		min(ns)	max(ns)	min(ns)	max(ns)	min(ns)	max(ns)	min(ns)	max(ns)
$t_{CLMRL}$	MRD active Delay	0	35	0	35	-45	38	-45	38
$t_{CLMRH}$	MRD inactive Delay	0	35	0	35	0	35	0	35
$t_{CLAV}$	ADDRESS valid Delay		128		78		78		68
$t_{CLAX}$	ADDRESS hold time	0		0		0		0	
$t_{DVCL}$	Read DATA set up time	54		40		38		28	
$t_{CLDX}$	Read DATA hold time	10		10		10		10	
$t_{MRLH}^{*1}$	MRD PULSE WIDTH	407 (2T)		376 (3T)		376 (3T)		305 (3T)*3	
$t_{IRLH}^{*1}$	IOR PULSE WIDTH	610 (3T)		501 (4T)		361 (4T)		392 (5T)	
$t_{CLTRL}$	IOR active Delay	14	74	80	138	-47	50	-47	50
$t_{CLIRH}$	IOR inactive Delay	14	74	14	74	0	35	0	35
$t_{YLC}^{*2}$	IORDY inactive set up	237		159					
$t_{YHCH}$	IORDY active set up	168		116		60		43	

(\*1) CPU に外部から Wait をかけない時

(\*2) IORDY はバスに対して非同期でよい、本規格値内でクロックに対して変化させれば、次の cycle の動作が保証される

(\*3) オプション ROM のアドレス空間では407(4T)となる

### ▶ I/Oライト・サイクル

I/Oライト・サイクルでは、 $T_2$ ステートの始まりにデータ・バスの内容が確定した後に、 $T_3$ の始まりで  $IOW_0$  がアクティブになります。その後、 $T_4$ ステートの始まりで  $IOW_0$  がインアクティブになり、十分なホールド・タイムを確保した後にデータ・バスのデータが不定になります。 $IOW_0$  がアクティブの間、書き込みデータは確定しており、前後のマージンも十分確保されています。

I/Oライト・サイクルにおいても自動的に挿入されるウェイト・サイクル数は、I/Oリード・サイクルと同様です。

以上のタイミングには、CPUが8086の場合でもV30の場合でも大きな違いはありません(図2-3参照)。問題は、 $IOW_0$  がアクティブになっている時間  $t_{IWLH}$  ですが、次のようにして求めることができます。

$$t_{IWLH} = (1+n) \times t_{cy} - t_{CLIL} + t_{CLIH} \dots\dots\dots(2)$$

最もタイミングが厳しくなるV30の8MHzモードでは、

$$t_{IWLH} = 3 \times 125.2 - 100 + 0 = 275(\text{ns})$$

になります。

### 〔3〕拡張ボードの設計例1 汎用ICによるI/Oポート

前節のスロット・バスの諸定義の解説は、すべての信号線について解説したためとデータがほとんどだったために、ビギナには難解であったかもしれませんが、しかし、実際に使用する信号線は限られており、またタイミング設計などもポイントさえ押さえればほとんどの場合に対応できます。

以下に示す設計例は、スロット・バス信号線の意味を理解し、拡張基板の基本的な設計方法を示すことが目的です。したがって、設計例の回路がすぐに何かに使えるといったものではありません。

#### 3-1 I/Oアドレスについて

どのようなI/O基板を設計する場合においても、共通に解決しなければならない事項として、**拡張基板の**

2000 2000 2000 2000 2000 2000  
ヤート TS TC TC TC TC TC



記号	パラメータ	5MHz mode(8086)		8MHz mode(8086)		8MHz mode(V30)		10MHz mode(V30)	
		min(ns)	max(ns)	min(ns)	max(ns)	min(ns)	max(ns)	min(ns)	max(ns)
$t_{CLMWL}$	MWC <sub>0</sub> active Delay	0	35	0	35	-45	38	-45	38
$t_{CLMWH}$	MWC <sub>0</sub> inactive Delay	0	35	0	35	0	35	0	35
$t_{CLIWL}$	IOW <sub>0</sub> active Delay	-70	74	-43	74	14	100	14	100
$t_{CLIWH}$	IOW <sub>0</sub> inactive Delay	14	74	14	74	0	35	0	35
$t_{CLDV}$	Write Data Valid Delay		122		72		78		68
$t_{CHDX}$	Write Data hold time	10		10		10		10	
$t_{CLEL}$	MWE <sub>0</sub> active Delay(CPU)	11	37	11	37		69		41
$t_{CLEH}$	MWE <sub>0</sub> inactive Delay(CPU)	19	65	19	65		120		34

じDOhでも、00DOhからFFDOhまでの256通りのアドレスが別々に使用可能なのです。

2500

- ① I/Oアドレスの下位は、とりあえずDOh~DFhの連続した空間に必要なバイト数だけ割り当てる。
- ② 他の基板との競合を避けるため、アドレスの上位8ビットもデコードして16ビットのアドレスを与える。
- ③ 上位アドレスは、ディップ・スイッチにより設定可能ようにする。

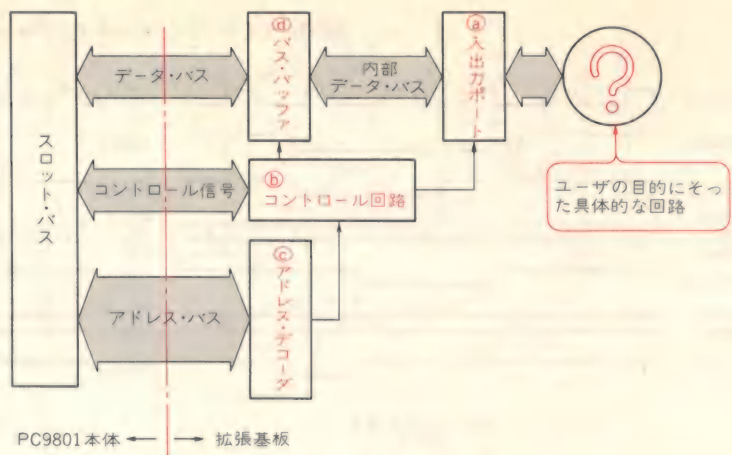
②

幸いなことに、正確にいうとPC9801にはもっと多くのI/Oアドレスが開放されているのです。なぜならば前述の空きアドレスは、上位アドレスを無視した場合のことで、実際にはそれぞれのアドレスについて、上位アドレスが00hからFFhまでの256通りの組み合わせが選択可能だからです。すなわち、例えば同

まず最初に、汎用IC(TTLなど)を用いたI/Oポート



図3-1 一般的なI/O基板の構成



を設計する場合の例をあげて、I/Oポートの意味や基本的な設計要領について詳しく解説します。

何を設計する場合にもそうですが、最初にこれから設計するものの仕様を明らかにしなければなりません。本設計例の場合、目標仕様は次のようなものとします。

- ①入出力点数はともに1ワード(2バイト)の16チャンネルとする。
- ②I/Oアドレスは入出力ともに××D E hおよび××D F h番地とする(××は任意に設定可能)。
- ③ポートに対するアクセスは、バイトおよびワードでのアクセスの両方可能とする。
- ④使用するICはLS TTLとする。

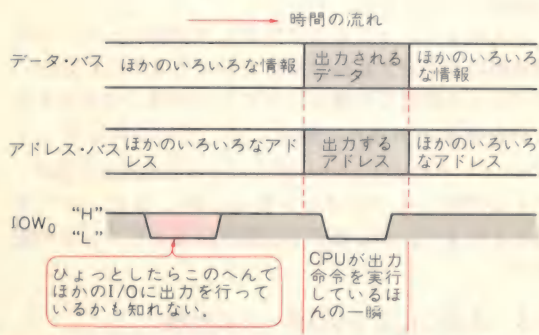
一般的なI/O基板の回路を大まかにブロック化すると、図3-1に示すような構成要素に分けられます。

①は実際に出力データをラッチ(記憶)して出力するための出力ポートや、入力データをバスに取り込むための入力ポート。

②は①の入出力ポートにストローブ・パルスを与えるためのコントロール回路。

③はCPUから所定のアドレスが出力された場合だ

図3-2 出力ポートの働き



CPUが出力命令を実行しても、ほんの瞬間に必要なデータがデータ・バスに出力されない。出力ポートは、自分のアドレスが指定されてIOW0が“L”レベルになったことを知ると、すかさずデータを取り込んでそれを保持しなければならない。

けに動作させるためのアドレス・デコーダ。

①は基板のファンイン(入力電流)やファンアウト(出力電流)の規定値を満足させるためのバス・バッファ。

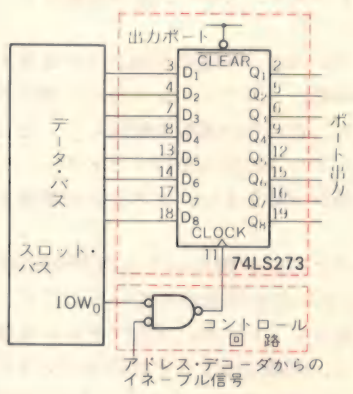
②の入出力ポートは、実際には専用の周辺LSIを用いる場合が多い(設計例2を参照)のですが、本設計例ではI/Oポートの構成方法を理解するために、あえて標準ICを用いて設計することにしました。

### 3-4 出力ポートの設計

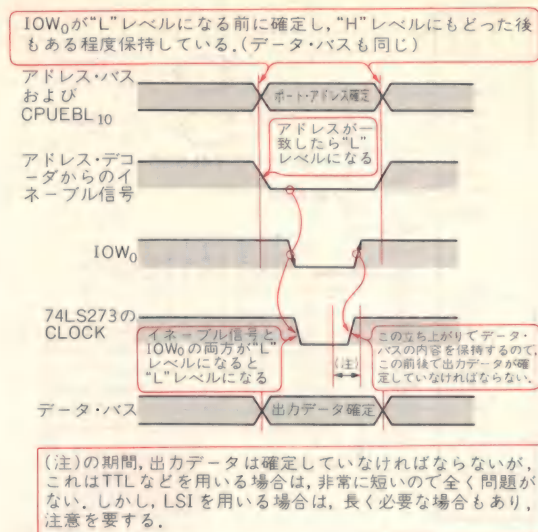
出力ポートの役目は、CPUが出力(O U T)命令を実行した瞬間にしかデータ・バスに出力されない出力データを、他の回路や人間が使用可能のように持続的な信号にしてやることです(図3-2)。実際には、この一瞬のデータをラッチまたはDタイプ・フリップフロップに記憶させます。

CPUが出力命令を実行すると、ポートのアドレスと出力データをバスに出力したのちに、IOW0(I/Oライト)を一瞬アクティブにします。IOW0はアクティブ“L”の信号線ですから、通常は“H”レベルで、

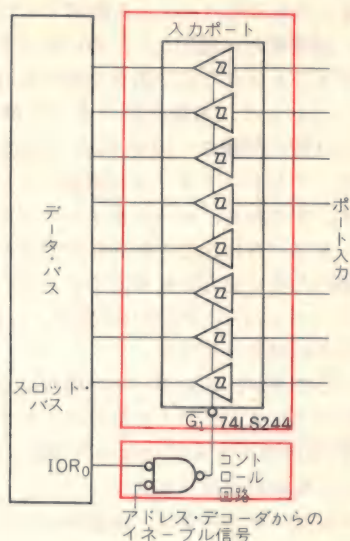
図3-3 出力ポートの構成



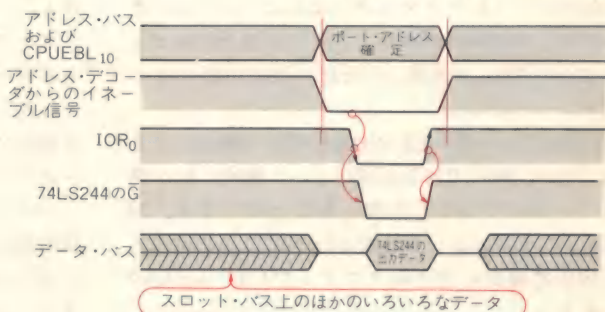
〈図3-4〉 出力ポートのタイミング



〈図3-5〉 入力ポートの構成



〈図3-6〉 入力ポートのタイミング



出力動作を行う場合に“L”レベルになります。“L”レベルになっている時間は機種やクロック周波数により異なりますが、最低でも約300nsは確保されています。

300nsという時間は人間にとってみるとほんの瞬間の出来事ですが、標準のロジックIC（TTLやHCMOSなど）にとっては十分余裕のある時間です。ただし、周辺LSIの中にはもう少し時間が必要なものもありますから、注意が必要です。

本設計例では16ビットの出力ポートが必要ですから、D-FFが8個入った74LS273を2個使用します（図3-3）。必要なポート数が少ない場合には、必要な数に応じて74LS74、LS175、LS174などを用いればよいでしょう。また、74LS374を用いると出力電流が大きく取れますので、LEDぐらいでしたらそのままドライブできます。

74LS273（他のD-FFも同じ）は、クロック入力の立ち上がり（ポジティブ・エッジ）でデータをラッチします。このクロックを作成するのが⑥のコントロール回路で、前述のIOW<sub>0</sub>を用いて作ります。IOW<sub>0</sub>がアクティブ（“L”レベル）になっている間とその前後では、出力されるべきデータが確定してデータ・バス上に出

力されています。

したがって、IOW<sub>0</sub>の立ち下がり（立ち上がり）のどちらを用いてもよいのですが、ここではIOW<sub>0</sub>の立ち上がりがクロックの立ち上がりになるようにします。

図3-4に出力ポートのタイミングを示します。

出力ポートにクロックが与えられるのは、自分に割り当てられたアドレスが指定された場合だけにしなければなりません。そこで、③のアドレス・デコーダで作成されたイネーブル信号を用いて、必要な場合にのみクロックが出力されるようにします。

### 3-5 入力ポートの設計

一方、入力ポートは外部の信号をCPUが入力（IN）命令を実行した瞬間にデータ・バスに取り込むための回路です。すなわち、IOR<sub>0</sub>（I/O R<sub>0</sub>）がアクティブ（“L”レベル）になっている間だけ、信号をデータ・バスに出力すればよいのです。これには、3ステート・バッファを用い、その出力イネーブル信号をIOR<sub>0</sub>を用いて作成します。

具体的には、図3-5に示したように8チャンネルの3ステート・バッファ74LS244を2個使用して16ビット



の入力ポートを構成することになります。筆者は、74LS244の入出力のピン配置があまり好きではないので、74LS245などの双方向バッファを流用することが多いのですが、消費電力が増加し、出力に常に1ゲート入力ぶら下がることになるのであまり勧められません。

入力ポートとして、本設計例のように標準ICを用いる場合には特に問題ないのですが、**周辺LSIを用いる場合には、アクセス・タイムが重要なポイント**になってきます。すなわち、ポートをイネーブルにしても即座にデータがバスに出力されるわけではなく、わずかな時間遅れた後に出力されるのです。**この遅れ時間がアクセス・タイムで、PC9801の場合、これが約280ns以下でなければなりません。**

当然のことですが、入力ポートの場合にも出力ポートと同様にアドレス・デコーダからのイネーブル信号とIOR<sub>0</sub>の両方がアクティブになった場合にのみ動作するようにしなければなりません。

③のアドレス・デコーダは、指定されたアドレスの場合にのみポートがアクティブになるようにするためのものです。本設計例の場合、アドレスの上位8ビット、すなわちAB<sub>081</sub>～AB<sub>151</sub>は任意に設定可能になっています。

この回路はEX-ORやEX-NORを用いて構成してもよいのですが、最近では74LS688という便利なICが発売されていますのでこれを用いることにします。74LS688は8ビットどうしの比較を行うICで、2組の入力が一致した場合に出力がアクティブ(“L”レベル)になります。

入力的一方をアドレス・バスに接続し、他方をプルアップするとともにディップ・スイッチを通してグラウンドに接続します。ディップ・スイッチがOFFの場合、プルアップにより入力は“H”レベルになり、ONの場合には当然“L”レベルになります。

### 3-6 アドレス・デコード部の回路構成

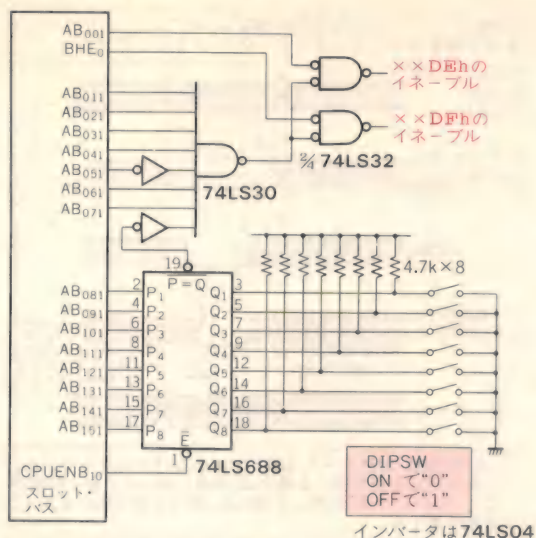
後述のAB<sub>001</sub>を除いた下位7ビットのアドレスは、DEhになるようにNANDゲートとNOTゲートでデコードします。

デコードする場合に忘れてはならないのが、CPUENB<sub>10</sub>という信号線です。この信号線がアクティブ(“L”レベル)な場合のみ、I/Oポートが動作するように設計しなければなりません。

本設計例のポイントのひとつは、ポートに対するアクセスがバイトでもワードでも可能であるという点です。これはメモリ基板を設計する場合には当然なことなのですが、I/O基板の場合には特に考慮されない場合がほとんどです。

というより、一般的にI/O基板は8ビットCPUの周辺LSI(8251A、8255Aなど)を用いて設計する場合は

〈図3-7〉アドレス・デコーダ



多く、**ワード・アクセスが原理的に不可能**なのです(データ・バスが8本しかないのでワードすなわち16ビットのデータを一度にアクセスすることはできません)。

しかし、本設計例のように汎用ICを用いて構成する場合には、このような制限はなく、バイトおよびワードでのアクセスを可能にするべきでしょう。

バイトおよびワードでのアクセスを実現するための信号線は、AB<sub>001</sub>(A<sub>0</sub>)とBHE<sub>0</sub>です。**名称は全く異なりますが、これら2本の信号線は同じような働きをします。**AB<sub>001</sub>は、1ワード(16ビット)のうち下位の8ビット(すなわち偶数アドレス)に対してアクセスが行われる場合にアクティブ(“L”レベル)になり、一方BHE<sub>0</sub>は上位の8ビット(すなわち奇数アドレス)に対してアクセスが行われる場合にアクティブ(“L”レベル)になります。

したがって、AB<sub>001</sub>とBHE<sub>0</sub>の両方がアクティブの場合には、上位バイトと下位バイトの両方に対してアクセスが行われること、すなわちワード・アクセスを意味します。

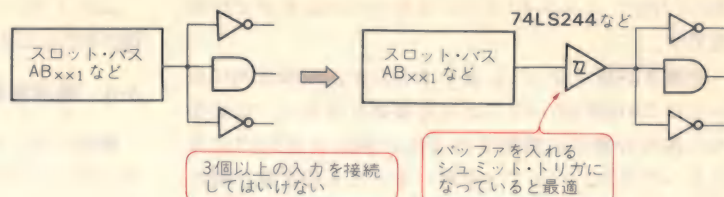
以上をまとめると、**偶数アドレスに接続したデバイスはAB<sub>001</sub>がアクティブ(“L”レベル)のときに動作し、奇数アドレスに接続したデバイスはBHE<sub>0</sub>がアクティブのときに動作するように設計すればよい**のです。

図3-7にアドレス・デコード例を示します。

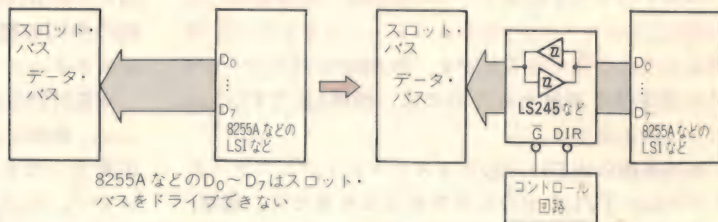
### 3-7 バス・バッファの考え方

④のバス・バッファの目的は、大きく図3-8に示したように二つに分けられます。一つは拡張基板の入力端子と基板内の回路の間に挿入する場合で、これは入力端子に流れる電流をスロット・バスの許容値内におさめることが目的です。

〈図3-8〉 バス・バッファが必要な場合



(a) 入力が3個以上並列になる場合(アドレス・バス、コントロール信号)



(b) 出力がバッファ・タイプでない場合(データ・バス)

TTL ICは入力端子に決まっただけの電流(ファンイン)を流さなければ動作しないのですが、いくつかのICの入力端子が並列に接続されると、この電流の合計値がスロット・バスのドライブ能力(ファンアウト)を超えてしまう恐れがあります。

具体的には、PC9801のスロット・バスはLS TTL (74LS×××という品番がついたシリーズ)の入力を2個までドライブする能力があります。したがって、2個を超える入力が並列になってしまう場合には、スロット・バスの端子と内部回路の間にバッファを入れなければならない。ただし、LS TTLの入力電流

値(ファンイン)はすべてが等しいわけではなく、例外的に異なる値を取ることもあります。

その場合は、1個までしかドライブできないこともあります。また、ここでいうバッファは、特にバッファ専用のICである必要はなく、通常のLS TTLに間に1段入れるだけでよいのですが、どうせ入れるなら入力がシュミット・トリガになっているIC(インバータであれば74LS14)を用いたほうがノイズに対して有効です。

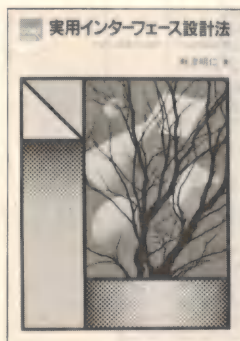
もともと、バッファ専用のICの多くは(74LS244な

## ▶▶▶ CORE BOOKS

CQ出版社

マイコン活用のためのハードウェア技術入門  
実用インターフェース設計法

畔津 明仁 著  
A 5 判・212頁  
定価 1,400円



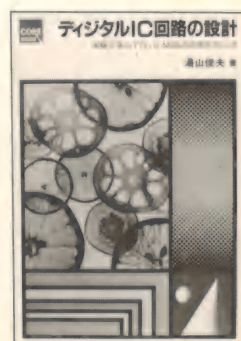
安定に、正確に、効率よくまわす技術  
DCモータの制御回路設計

谷腰 欣司 著  
A 5 判・200頁  
定価 1,500円



実験で学ぶTTL、C-MOSの応用テクニック  
ディジタルIC回路の設計

湯山 俊夫 著  
A 5 判・256頁  
定価 1,600円





ど)入力がシュミット・トリガになっていますので好都合です。

前置きが長くなってしまいましたが、本設計例においてはこの目的のバッファは必要ありません。なぜなら、各入力端子に接続されるICの数は2個以内におさまっており、バッファを用いなくても規定値を満たすからです。

バス・バッファのもう一つの目的は、**拡張基板の出力電流(ファンアウト)をスロット・バスをドライブ可能のように増強すること**です。PC9801のスロット・バスの規定値を満足するためには、標準LS TTLの出力では力不足です。

拡張基板の出力は、必ず3ステート・バッファ・タイプのLS TTLで出力しなければなりません。通常、拡張基板から出力される信号線はデータ・バスだけですが、本設計例では入力ポートの出力(74LS244)が3ステート・バッファになっていますから、この他にバッファを入れる必要はありません。

しかし、**入力ポートとして周辺LSIを用いる場合などには、バッファが必要になってきます**。通常、データ・バスは双方向(入力にも出力にもなるという意味)で使用されるのが一般的ですから、この場合には双方向バス・バッファ(74LS245など)を用いるのが便利です。

このようにして設計された拡張基板の全体回路図を、図3-9に示します。

### 3-8 電源容量も要注意

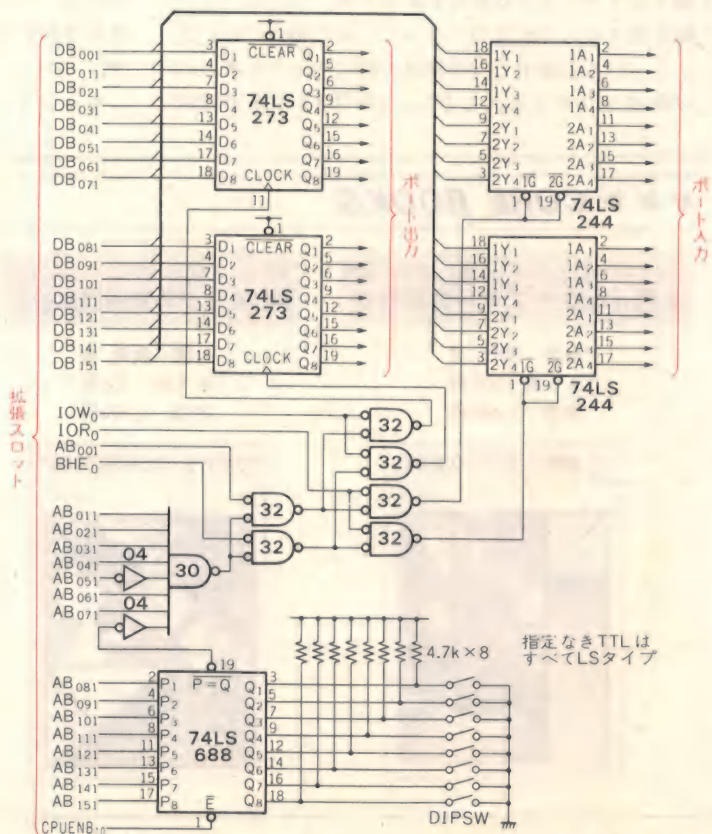
最後になりましたが、**拡張基板を設計するうえで案外見落としがちなのが電源容量の問題**です。PC9801のスロット・バスは、+5Vの電源容量を1スロット当たり0.5A以内に規定しています。でき上がった回路の各ICの電源電流の最大値の合計が、規定値以内におさまっているかどうかの検証が必要です。

本設計例程度の回路では電源容量の心配はありませんが、複雑な回路になると、電源容量が不足する場合があります。そういった場合は、LS TTLを用いずに、**ALS TTLやHC-MOSを使用するとある程度電源容量に余裕ができます**。

ALS TTLは、LS TTLに比較してより高速で、低消費電力なので完全に置き換えが可能です。HC-MOSはC-MOS構造になっているために、静的消費電流はほんのわずか( $\mu$ Aのオーダー)ですが、LS TTLと完全に互換性があるわけではありません。

特に、高速性を要求する場合や、出力電流(ファンアウト)を多く必要な場合には注意が必要です。実際問題として、HC-MOSの3ステート・バッファ出力で、PC9801のバスをドライブするには多少無理が

〈図3-9〉 設計例1の全回路



あるようです。

#### 4 拡張ボードの設計例 2 80系周辺LSIの接続

設計例1ではスロット・バスの基本的な使用方法を解説するために、標準ICを使用して設計を行いました。実際には専用周辺LSIを用いたほうが有利な場合が多いでしょう。

8086は8ビットの老舗である8080Aから発展した16ビットCPUですから、80系のペリフェラルLSIを用いるのが順当です。

ここでは、80系ペリフェラルLSIの代表である8255A(パラレル・インターフェース)を接続する場合を例にあげて解説します。他の80系LSIもほとんど同じ考え方で接続可能です。まず、目標仕様を次のように設定します。

- ①8255Aを4個使用し、96ビットのI/Oポートを構成する。
- ②I/Oアドレスは、 $\times\times D0h$ から $\times\times DFh$ までの16バイトとする( $\times\times$ は任意に設定可能)。

##### 4-1 8255Aの内部レジスタの割り付け

基本的な考え方は設計例1で解説しましたので、ここでは設計例1と異なる点について解説することになります。

8255Aの内部には4バイト分のレジスタがあり、それらのどれにアクセスするかを選択するのが8255Aの $A_0$ 、 $A_1$ という入力端子です。8ビットCPUに接続する場合には、通常この2本をアドレス・バスの最下位、すなわち $A_0$ と $A_1$ に接続すればよいのですが、16ビットCPUである8086の場合はそう単純にはいきません。

8086のデータ・バスは16ビットあり、偶数番地のデータは必ずデータ・バスの下位8ビットを用いてアクセスされ、奇数番地のデータは上位8ビットを用いてアクセスされます。一方、8255Aのデータ・バスは当然8ビットですから、それを例えば下位8ビットに接続した場合には、偶数番地しか利用できないことになります。

したがって、この場合アドレス・バスの最下位 $AB_{001}$ は偶数番地(すなわちデータ・バスの下位ビット)に接続された8255Aが選択されたことを示すことに用いられ、内部レジスタを選択するためには用いることができません。つまり、 $A_0$ と $A_1$ に接続できるのは $AB_{001}$ を除いた下位2ビットの $AB_{011}$ と $AB_{021}$ なのです。

本設計例では8255Aを4個接続しますから、全体で $4\times4=16$ バイトのI/O空間を使用することになります。しかし、その割り当ては個々の8255Aに対して連続ではなく、表4-1のように1バイトおきの割り当て

表4-1 I/Oアドレスと8255Aの割り当て

I/Oアドレス	$AB_{031}$	$AB_{001}$	$BHE_0$	8255A	内部レジスタ
$\times\times D0$	0	0	$\times$	#0	ポートA
$\times\times D1$	0	$\times$	0	#1	ポートA
$\times\times D2$	0	0	$\times$	#0	ポートB
$\times\times D3$	0	$\times$	0	#1	ポートB
$\times\times D4$	0	0	$\times$	#0	ポートC
$\times\times D5$	0	$\times$	0	#1	ポートC
$\times\times D6$	0	0	$\times$	#0	コマンド
$\times\times D7$	0	$\times$	0	#1	コマンド
$\times\times D8$	1	0	$\times$	#2	ポートA
$\times\times D9$	1	$\times$	0	#3	ポートA
$\times\times DA$	1	0	$\times$	#2	ポートB
$\times\times DB$	1	$\times$	0	#3	ポートB
$\times\times DC$	1	0	$\times$	#2	ポートC
$\times\times DD$	1	$\times$	0	#3	ポートC
$\times\times DE$	1	0	$\times$	#2	コマンド
$\times\times DF$	1	$\times$	0	#3	コマンド

$\times$ は0,1に無関係なことを示す

になります。

この割り当てを行うのが、 $AB_{001}$ 、 $BHE_0$ 、 $AB_{031}$ の信号線です。これら3本の信号線とアドレス・バスの上位をデコードしたイネーブル信号の組み合わせによって、個々の8255Aに対するチップ・セレクト信号(CS)を作成します。

80系の周辺LSIにおいて、アクセスのストロープは $\overline{RD}$ と $\overline{WR}$ により行われます。このことは、8086を使用しているPC9801でも同様で、 $\overline{IOR_0}$ を $\overline{RD}$ に接続し、 $\overline{IOW_0}$ を $\overline{WR}$ に接続するだけで完了です。

8255AにはRESET入力端子があり、これをバスの $\overline{RESET_0}$ に接続すればよいのですが、この場合 $\overline{RESET_0}$ がアクティブ“L”なのに対してRESETはアクティブ“H”ですから、インバータを介して接続します。

##### 4-2 8255Aにはバス・バッファが必要

設計例1の場合、特別なバス・バッファは必要ありませんでしたが、本設計例ではそれが不可欠です。8255Aのデータ・バス出力はスロット・バスの規定値をそのままドライブすることはできません。

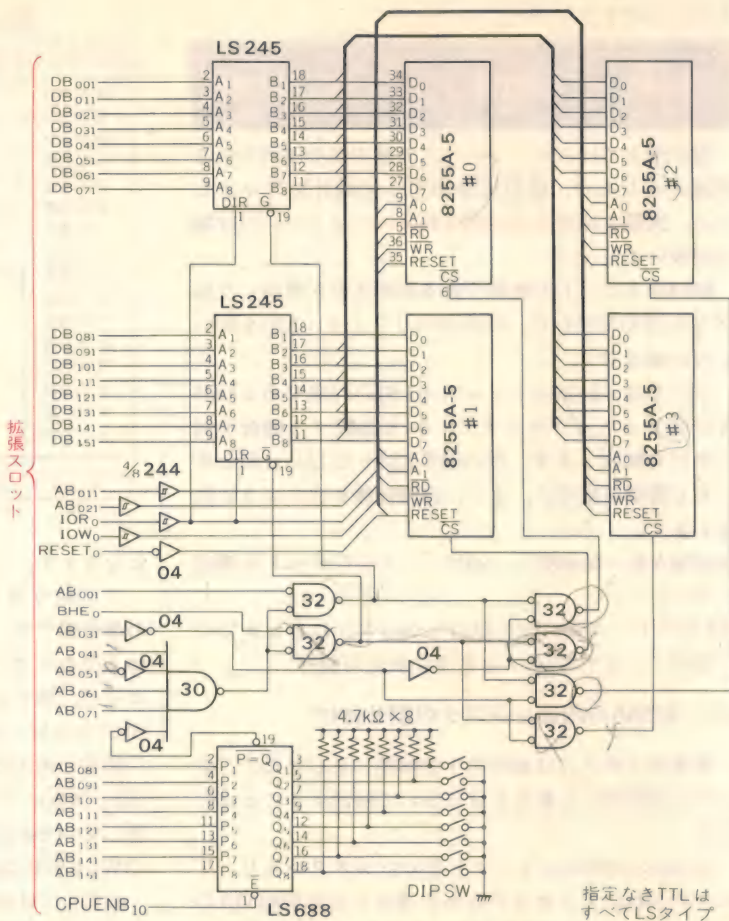
したがって、双方向バス・バッファ74LS245を用いてバッファリングします。このとき、重要なのが方向(DIR)制御の問題です。

ここでは、バッファの方向を通常はスロット・バスから8255Aの方向に向けておき、読み出しを行う場合、すなわち $\overline{RD}$ がアクティブになった場合のみに、逆方向に転換するという方法をとります。また、バッファのイネーブル端子には、アドレス・デコードの出力を入力して、所定のアドレスに対してアクセスされた場合にのみ、バッファが動作するようにします。

8255AはNMOSのLSIですから、入力端子( $A_0$ 、 $A_1$ 、 $\overline{RD}$ 、 $\overline{WR}$ など)の直流的な入力電流はごくわずかです。



〈図4-1〉 設計例2の全回路



しかし、入力容量は比較的大きく、本設計例のように4個も並列に接続する場合にはやはりバッファリングを行うのが賢明でしょう。

### 4-3 タイミング設計の考え方

設計を行う場合のポイントのひとつにタイミング設計がありますが、本設計例の場合はかなり厳しくなります。最大のポイントは、**RDおよびWRのパルス幅が8255Aの規定値を満たすかどうか**です。RD、WRの規定値は、高速タイプの8255A-5の場合でも、どちらも最小300ns必要です。スロット・バスのIOR<sub>0</sub>(すなわちRD)のパルス幅 $t_{IRLH}$ の最小値は次式で求められます。

$$t_{IRLH} = (1 + n) \times t_{cy} - t_{CLIRL} + t_{CLIRH}$$

最も厳しくなるVシリーズ8MHzの場合に、

$$3 \times 125.2 - 50 + 0 \approx 326\text{ns}$$

となり、ワースト・ケースでも規定値を満たすことができます。

ところが、IOW<sub>0</sub>(すなわちWR)のパルス幅 $t_{IWLH}$ の最小値は、**Vシリーズの8MHzで275ns**となり、**厳密には規定値を満たさない**ことになります。したがって、このままの設計ではVシリーズの8MHzでは動作が保証されません。すべての機種およびクロック周波数で動作を保証するためには、さらにウェイトを入れてパルス幅を延ばさなければなりません。

もっとも、上記の最小値は十分マージンをもったうえでの最悪値だと考えられ、また、8255A-5の要求値についてもかなりのマージンが含まれているはずですが、実際問題としてはすべての場合で動作することでしょう。このようにして設計した全体回路を図4-1に示します。

# PC9801用1Mバイト・メモリ・ボードの作り方

本章では、PC9801用のメモリ・ボードの作り方およびRAMディスクとしての活用法について解説します。現在では、市販のボードも安く購入できますが、パソコンとのメモリ・インターフェースを知るためには大変役に立ちます。

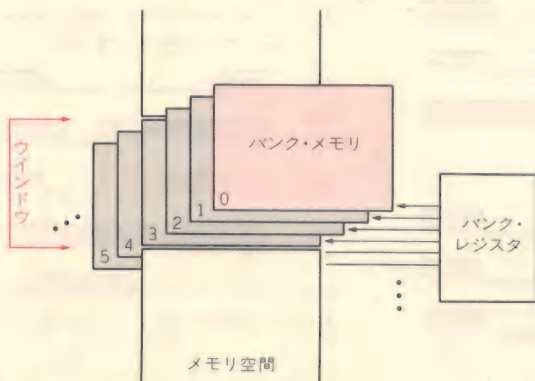
PC9801シリーズのインターフェース・タイミングの実例として、256KのDRAMを用いたメモリ・カードを設計してみました。最近ではPC9801用のオプション・カードの価格が低下し、ほとんどRAMの値段でインターフェース付きの基板が買えるようになっていきますので、コストの点ではあまりメリットはないのですが、標準的なバス・インターフェースとメモリ・サイクルの実際を知る点で意味があると思います。

## ① PC9801用1MバイトDRAMボードの設計

PC9801用の1MバイトDRAMボードは大体スタイルが固まっています。つまり、**PC9801がRAM領域として640Kバイトだけをユーザに開放している**ので、それを越える部分については、図1-1に示すようなバンク・スイッチ方式によって、あるウィンドウを通してアクセスする方式です。つまり、簡単なMMU(メモリ管理機構)をボード内に埋め込むわけです。

市販されているメモリ・ボードも大体これと同じ方法で、640Kバイトを越えるRAMボードをアクセスします。こうするとプログラム中で640Kバイトを越える領域をアクセスする場合に、**いちいちバンク切り替え(I/O命令になる)を行わなければならない**不便なのですが、現在のPC9801の使われ方では640Kバイト以上を必要とするアプリケーションはそう多くはなく、む

〈図1-1〉バンク・スイッチ方式のメモリ管理



しろ**高速のディスクの替わりに余ったメモリの領域を使用すること(RAMディスク)**が多いので、このような方式でも十分実用になっています。

ここではRAMディスクに対応できるように、アドレス変換機構を組み込むことにします。

### 1-1 1MバイトDRAMボードの仕様

DRAMを使用するメモリ・ボードの主眼点は以下のようになります。

- ① 要求されるアクセス・タイム( $t_{acc}$ )を満足できるか？
- ② 最短のメモリ・サイクル( $t_{cy}$ )を満足できるか？
- ③ リフレッシュ・タイミングはCPUから与えられるか、それはスペックを満たすか？
- ④ エラー対策を組み込むか？
- ⑤ インターフェース・ラインのファンアウト/ファンインは大丈夫か？

PC9801シリーズの内部は8086-2と8288の標準CPUインターフェースですから、これらについて8086-2、8288のカタログから必要な値を取りだして、必要なパラメータを計算してみます。

### 1-2 リード・モードのタイミング

まず、アクセス・タイムですが、READモードのクリティカルなタイミングは、 $\overline{RAS}/\overline{CAS}$ のクロックとして使われる  $\overline{MRC}_0$  (8288では  $\overline{MRDC}$ ) の立ち上がりからデータのセットアップまでの時間です。

外部ウェイトなしの場合、この時間は8MHzクロックで  $3t_{cy} - t_{CLML} - t_{DVCL}$  で  $375 - 35 - 20 = 320\text{ns}$  となります。5MHzでは内部ウェイト・サイクルがなくなり、 $2t_{cy} - t_{CLML} - t_{DVCL}$  で  $400 - 35 - 20 = 345\text{ns}$  となります。

つまり、その他のゲート遅延を0とした場合に320nsのアクセス・タイムが要求されているわけです。実際にはバス・バッファ、RAS/CAS作成のロジック、さらに容量遅延が加わるので、それらの合計をかりに70nsとすれば、 $t_{acc} < 250\text{ns}$ のRAMであれば使用できることになります。



これは現在の256KのDRAMの実力から見ると随分ゆるい規格です。 $t_{acc}$ が100ns程度のDRAMであればウェイトなしでも間に合うはずなのですが、PC9801側で強制的に1ウェイト(8MHzモード時)入ってきますので、これは確認できません。

### 1-3 ライト・モードのタイミング

同じくWRITEモードについて見ると、 $MWC_0$  (8288ではAMWC)のパルス幅がほとんどアクセス・タイムとなりますので、これはREADモードよりさらに緩くなります。

しかし、書き込みを図1-3(a)に示すアーリー・ライト・モードで行おうとすると、データの確定と $MWC_0$ の変化のタイミングがクリティカルなので注意が必要です。

DRAMのアーリー・ライト・モードは、 $\overline{CAS}$ の立ち下りの時点で $\overline{WE}$ とデータが確定していなければなりません。 $\overline{CAS}$ は、 $MWC_0$ すなわち $\overline{RAS}$ を遅延線によって一定時間(100ns)遅らせたものを用いますが、データの確定が $\overline{CAS}$ に先行していなければならず、 $MWC_0$ のディレイとデータ確定のディレイの差が問題になります。

AMWCは8288がクロックと同期をとって出しますが、 $AD_0 \sim AD_{15}$ を8086が出しますのでディレイ・タイムにかなり差があります。AMWCの最大遅延( $t_{CLML}$ )は、最小が10ns、最大が35nsであるのに対し、 $AD_0 \sim AD_{15}$ のそれは最大60ns(8086では110ns)となっています。

つまり、8086-2では $MWC_0$ から $AD_0 \sim AD_{15}$ のセッ

トアップまで最大50ns、8086では100nsとなります。遅延線の精度、安定性さらにゲート遅延のばらつきを考慮すると少し不安です。

そのためかNEC純正のDRAMボードはアーリー・ライト・モードを用いておらず、 $MWE_0$ をDRAMの $\overline{WE}$ に接続したリード・モディファイ・ライト・モードを用いた設計となっています。アーリー・ライトを用いてより安全なタイミングを作る方法として、WRITEモードの $\overline{CAS}$ を $MWE_0$ から作成する方法が考えられます。

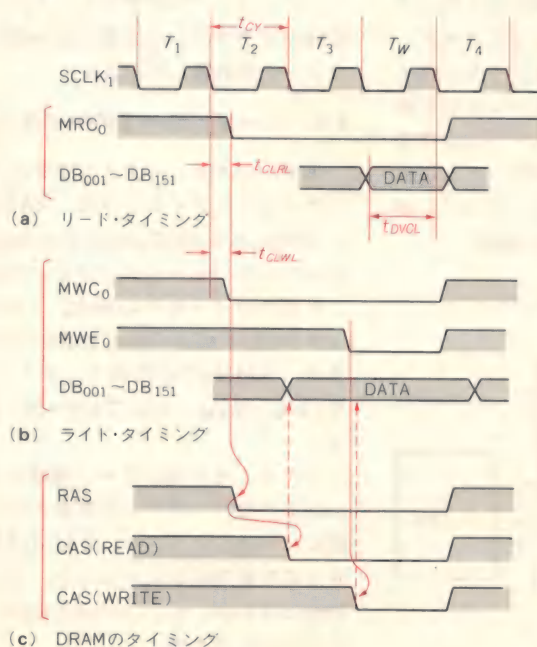
$MWE_0$ は $AD_0 \sim AD_{15}$ よりも明らかに遅く“L”になりますから、クリティカルな遅延時間に頼る必要はありません。当然、 $\overline{CAS}$ の幅は短くなりますが、 $MWE_0$ は最終のサイクルを表していますから、5MHzで200ns、8MHzでは125nsが保証されますので、DRAMの規格(60ns程度)と比べて十分すぎる余裕があります。

サイクルに関しては8086の最短メモリ・サイクルは4 $t_{cy}$ であり、しかも8MHzではウェイトが挿入されますから8MHzでも625nsであり、全く問題はありません。

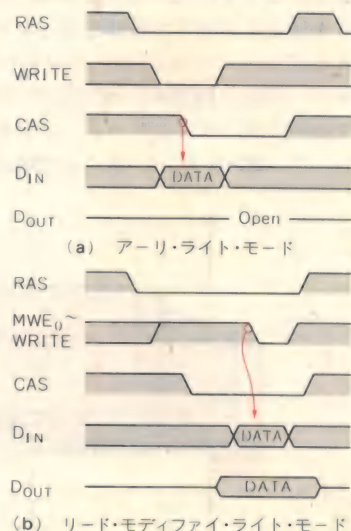
### 1-4 リフレッシュ動作について

DRAMでは記憶保持のためのリフレッシュ動作が必要です。PC9801内部にもDRAMが使用されているため、本体部にタイマとDMAコントローラを使ったリフレッシュ回路が内蔵されており、そのタイミングを借用できます。問題となるのは、リフレッシュ・アドレスの幅とサイクルです。PC9801も内部にDRAM

〈図1-2〉  
メモリ・アクセスの  
タイミング図



〈図1-3〉 DRAMの二つの書き込み方式



を使用していますし、バスを調べるとリフレッシュ・アドレスはAB<sub>001</sub>からAB<sub>081</sub>の8本出ています。

リフレッシュ間隔は、VMで約16 $\mu$ sに一つのロウ・アドレスがセレクトされていますから、256K DRAMの256リフレッシュ・サイクル/4msを満足できます。

外部メモリ・ボードに対してリフレッシュ・ボードであることを示すRFSH<sub>0</sub>が拡張バスに出ていますので、これを用いてRASオンリ・リフレッシュ・サイクルのタイミングを作成すればよいでしょう。

## 1-5 アドレス空間の割り当て

PC9801シリーズのRAM用のメモリ空間は640Kバイトしかありませんし、本体に実装されているRAMの容量も図1-4に示すように機種によって異なります。このため、本DRAMボードでは、なるべくいろいろな機種とメモリ構成に対応できるように1Mバイトのアドレス空間の割り当てを考えます。

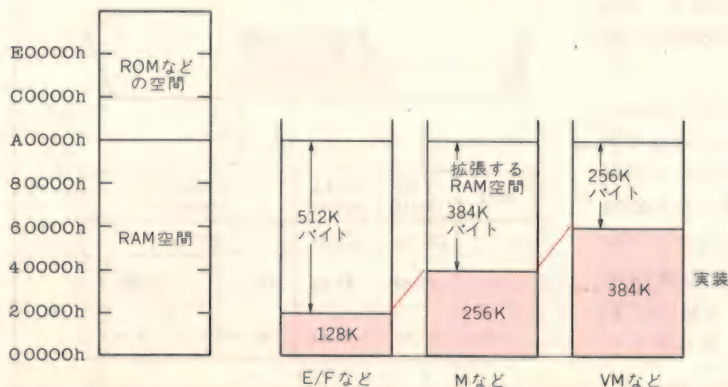
対応可能な構成は、すでに実装されているRAM容量が128Kバイト、256Kバイト、384Kバイト、512Kバイトで、しかも80000h番地から9FFFFh番地までの128KバイトにRAMが実装されていない場合です。このときに、未実装のメモリ空間にRAMを実装し、余ったメモリは80000h番地から9FFFFh番地に128Kバイト単位で切り替えてマッピングできます(図1-5)。

さらに同じボードを2枚、3枚、4枚と追加することにより、それらのメモリをすべて80000h番地からの128Kバイトにマッピングできるようになっています。

RAMボード上の1Mバイトのアドレス空間は、128Kバイト単位で8バンクに分割されます。このバンクをI/O空間にある8ビットのバンク・レジスタの値によって切り替え、80000h番地からの128Kバイトにマッピングします。

また、20000h番地から7FFFFh番地の間

〈図1-4〉 PC9801シリーズのメモリ空間と実装量の違い



のRAM未実装の空間に拡張RAMとして固定して割り当てるために、PC9801のメモリ空間を128Kバイト単位にデコードし、バンク・レジスタとは別に、RAMボードの上位から128Kバイト単位でRAM空間に割り当てられるようになっています。

拡張RAMとしての固定の割り当てアドレスは、回路図のジャンパJ<sub>1</sub>、J<sub>2</sub>、J<sub>3</sub>により制御され、

▶ J<sub>1</sub>によりバンク7を60000h番地からの128Kバイト

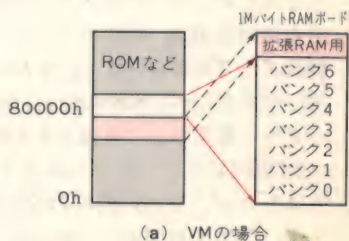
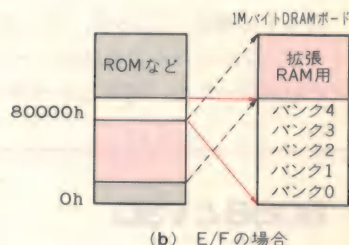
▶ J<sub>2</sub>によりバンク6を40000h番地からの128Kバイト

▶ J<sub>3</sub>によりバンク5を20000h番地からの128Kバイト

のように、それぞれ割り当てることができます。

バンク・レジスタは回路図上ではI/O空間のECh番地にありますが、偶数アドレスであれば容易に変更可能です。バンク・レジスタはリセット後にゼロになっていますので、ジャンパJ<sub>4</sub>がショートしていればバンク0が80000h番地からの128Kバイトに見え

〈図1-5〉 PC9801と拡張メモリ・ボードの関係



〈図1-6〉 2枚のボードを使用する場合のバンク割り当て

バンク7	バンク15	拡張RAMに使用 (J <sub>1</sub> ~J <sub>3</sub> を適当にショート)
バンク6	バンク14	
バンク5	バンク13	
バンク4	バンク12	
バンク3	バンク11	
バンク2	バンク10	
バンク1	バンク9	
バンク0	バンク8	

1枚目 (J<sub>4</sub>ショート)      2枚目 (J<sub>5</sub>ショート)



ています。

ボードを1枚のみ使用する場合は $J_4$ をショートし、必要に応じて $J_1$ 、 $J_2$ 、 $J_3$ をショートします。複数のボードを使用する場合は、1枚目は $J_4$ を、2枚目は $J_5$ を、3枚目は $J_6$ をそれぞれショートします。

図1-6に示すように、この状態では1枚あたり8バンクずつ、1枚目はバンク0からバンク7、2枚目はバンク8からバンク15という順序になり、バンク・レジスタに書き込む値で対応するボードのバンクが割り当てられます。拡張RAMとして固定で使用する部分は、一番最後のボードの $J_1$ 、 $J_2$ 、 $J_3$ により割り当てられることでバンクの連続性が保たれます。この場合、他のボードの $J_1$ 、 $J_2$ 、 $J_3$ はオープンにする必要があります。

## 1-6 タイミング設計

タイミングは $MRC_0$ 、 $MWC_0$ 、 $MWE_0$ から作ります。 $MRC_0$ と $MWC_0$ から $RAS$ を作り、 $RAS$ をディレイ・ラインで遅らせてアドレス・マルチプレクスの切り替え、さらにリードのときの $CAS$ を作ります。

ライトの場合の $CAS$ は、前に述べたようにディレイ・ラインから作ると本体のデータの確定からのマージンが少ないため、 $MWE_0$ から作っています。

この場合、 $CAS$ はリードと同じにして、 $WE$ を $MWE_0$ から作り、リード・モディファイ・ライト・サイクルで動作させれば、PC9801本体のDRAMとほぼ同じ動きになるでしょうが、**アーリ・ライトのほうがDRAMまわりの配線の引き回しが容易になり、データ・パッファも簡単化できるので、ここではアーリ・ライトを使用しています。**

リフレッシュは、本体から得られる $RFSH_0$ に合わせて $RAS$ を作っています。たいていのDRAMの設計ではリフレッシュが最も重要な問題となりますが、PC9801シリーズでは本体に依存する限り問題は起きません。

## ②ハードウェアの設計

以上を考慮して設計したDRAMボードの全回路図を図2-1に示します。次に、このボードの各部について説明します。

### 2-1 アドレスのデコード

PC9801のアドレス・バスの $AB_{171}$ 、 $AB_{181}$ 、 $AB_{191}$ の3本をデコードすると、そのアドレス空間との対応は図2-2のようになります。LS138の $Y_4$ がセレクトされ

## 8086とV30

V30は日本電気が独自路線のCPUとして開発したのですが、アーキテクチャは8086を踏襲しており、さらに命令追加、8080モード、サイクルの短縮など機能アップされています。また、ピン配置が8086と同一であるため、8086のために設計された基板に、パターン変更なしで実装できる特徴をもっています。

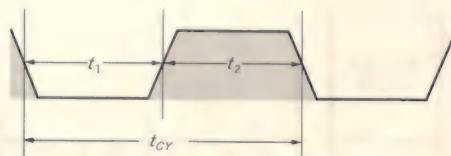
PC9801VシリーズからはメインCPUがV30になりましたが、それ以前のPC9801の機能アップのために、CPUを8086からV30に切り替えたいという要求があります。基本的にプラグ・コンパチブルなので、PC9801のCPUソケットから8086を抜き、V30を差し込めばよいのですが、V30のAC特性が一部8086-2と異なるので注意が必要です。

決定的な差は、**クロックのデューティ比が8086では1:2なのに対し、V30は1:1だという点です**(図A参照)。つまり、8086-2に8MHzクロックが8284Aから供給されている場合、その高レベル時間( $t_{2min}$ )は $(125/3 + 2)ns$ 、すなわち43nsなのですが、V30(8MHz)の $t_{2min}$ は50nsです。5MHz動作時の8284Aからの $t_2$ は68nsありますから、5MHzで動作させるという条件付きならばV30に差し換えて

も問題はありません。

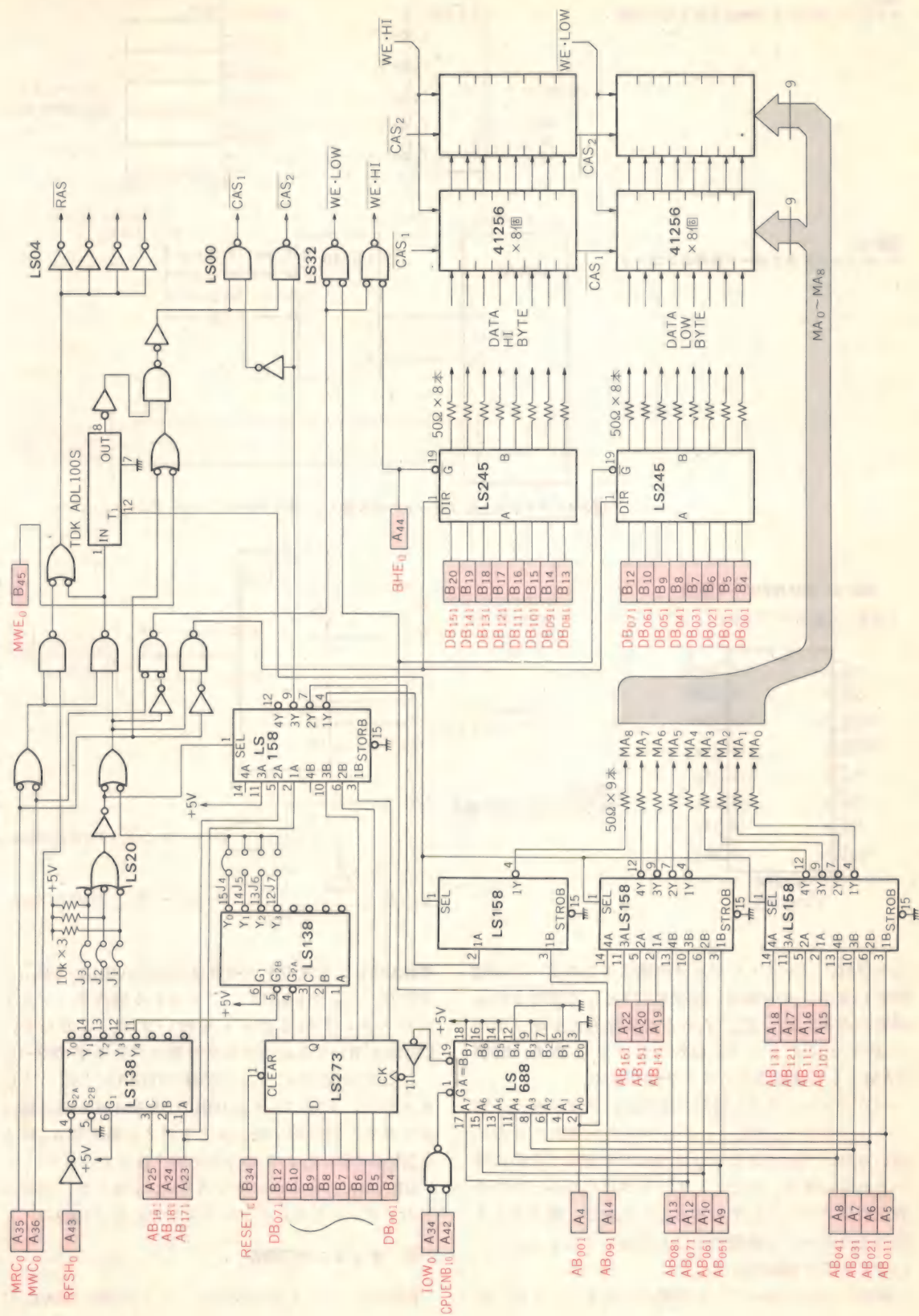
しかし、V30には10MHz版があります。その資料は入手していないので厳密な判断はできませんが、おそらく $t_{2min}$ は40ns程度でしょうから、それならば多分スベックをクリアできるものと思います。この $t_2$ のスペックは、あくまで最悪値ですから、実際には8MHzのPC9801E/FにV30の8MHz版を差ししても大体は動作します。しかし、悪条件下、例えば高温、電源不安定といった条件の元では動作を保証できません。

〈図A〉 V30のクロック規格と8284Aの規格



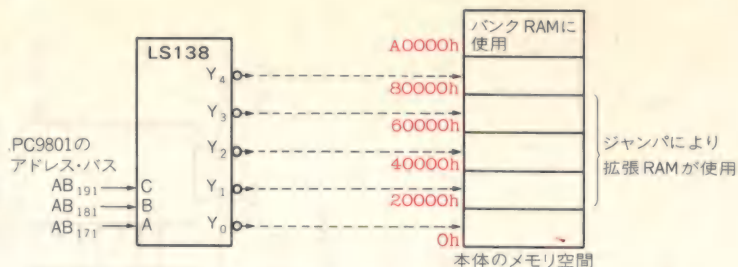
項目	V30 (8MHz)	8284A (8MHz)	8284A (5MHz)
$t_{cy}$	125 ns	125 ns	200 ns
$t_1$ (min)	60 ns	83 ns	$133 ns (\frac{2}{3} t_{cy} - 15ns)$
$t_2$ (min)	50 ns	43 ns	$68 ns (\frac{1}{3} t_{cy} + 2 ns)$

〈図2-1〉 1 Mバイト DRAMボードの全回路

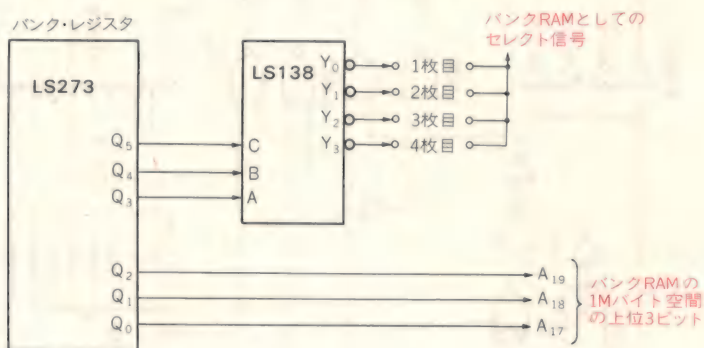




〈図2-2〉  
アドレス・デコードと対応するメモリ空間



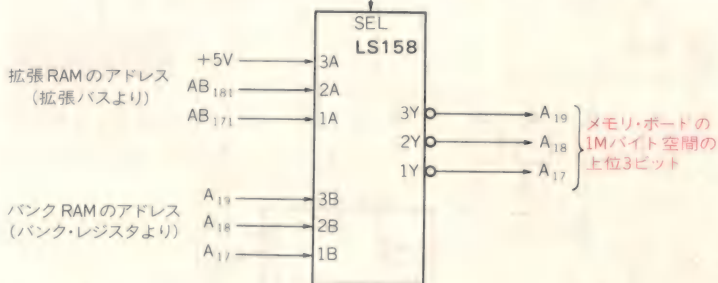
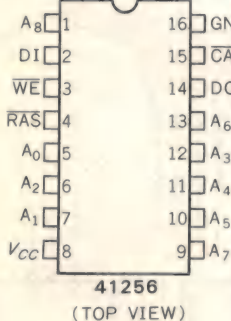
〈図2-3〉  
バンク・レジスタとボード番号のデコード



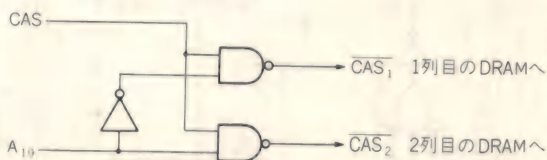
〈図2-4〉 アドレス上位3ビットの切り替え 拡張RAMのアドレス・デコードより

〈図2-5〉 DRAMのピン配置図

注意！一般のデジタルICと逆



〈図2-6〉  
列のCASによる切り替え



たときは、バンク・レジスタの値により本ボードは制御されます。その他に、拡張RAMとして使用される場合のため、 $Y_1$ 、 $Y_2$ 、 $Y_3$ から必要なものをジャンパによりセレクトして、ORのゲートに集め、拡張RAMとして使用されることを知ります。

バンク・レジスタLS273の出力は、**下位3本が1Mバイトの中から128Kバイトのバンクを選択するの**に用いられ、**上位の5ビットはボードを選択するの**に用いられています。ただし、ゲートが足りなかったため最上位はデコードしていません。しかし、最大でも6枚以上のボードを使用することはないでしょうからこれで十分です(図2-3)。

本ボードの1Mバイトの空間の上位3ビットは、拡

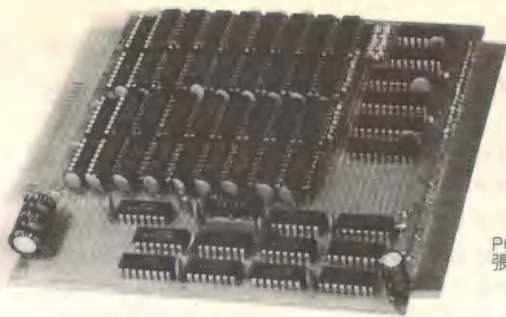
張RAMとして使用する場合は本体の $AB_{181}$ 、 $AB_{171}$ から作り、バンクRAMとして使用する場合はバンク・レジスタの下位3ビットを用います。このため、LS158を用いてこの3本を切り替えています(図2-4)。

ここでは図2-5に示した256KのDRAMを用いていますから、1Mバイトは16個のDRAMが2列必要になります。列の切り替えはアドレスの最上位 $A_{19}$ により図2-6に示したようにCASを切り替えています。

DRAMには9本のアドレス・ピンがあって、18ビットのアドレスをマルチプレクスしています(図2-7)。

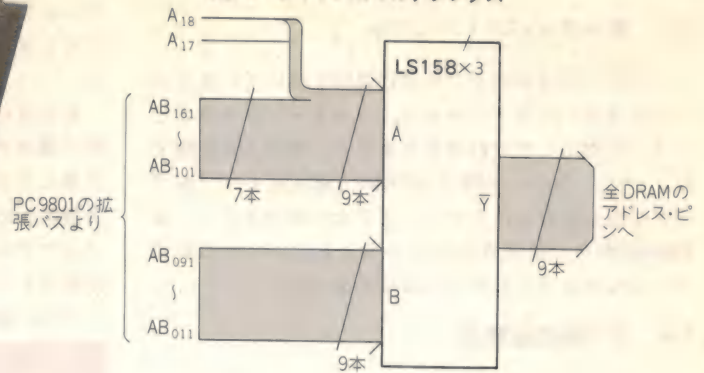
## 2-2 タイミング回路

RASは、リード時の $MRC_0$ とライト時の $MWC_0$ の

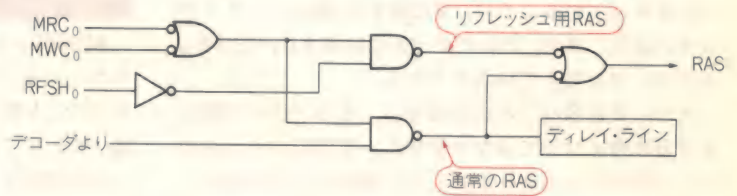


〈写真2-1〉 製作した1MバイトDRAMボード

〈図2-7〉 アドレスのマルチプレクス



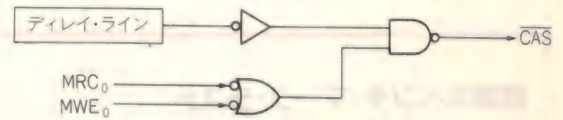
〈図2-8〉 RAS発生回路



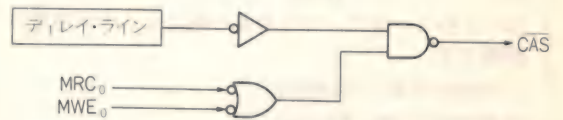
ORを用います。ただし、メモリ・リフレッシュのためにRFSH<sub>0</sub>ともORをとります(図2-8)。RASはたくさんのDRAMに接続されますので、LS04を4個用いて8個ずつのDRAMのRASに接続します。

CASは、リード時にはディレイ・ラインを用いて出力を遅らせますが、ライト時はMWE<sub>0</sub>のライト・ストロブを用いています(図2-9)。このため、ライト時にも十分なタイミングの余裕があります。CASはA<sub>19</sub>を用いて2列のDRAMに切り替えを行っています。ここで使用したディレイ・ラインの仕様を図2-10に示します。

〈図2-9〉 CAS発生回路

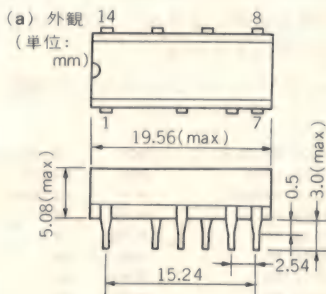


(a) リード時のCAS

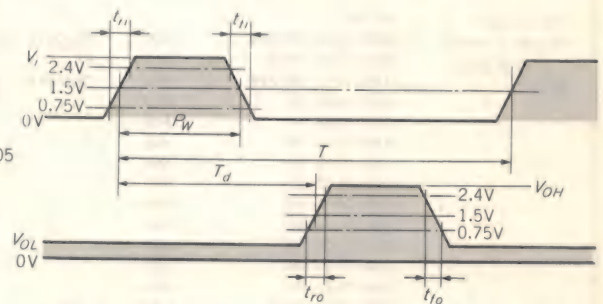


(b) ライト時のCAS

〈図2-10〉 ADL100S(TDK株)の仕様



品 名	全遅延時間	タップ間遅延時間	立ち上がり時間
ADL100S	100ns ± 5%	20 ± 3ns	4ns (max)



$V_i$  : 入力電圧  
 $T_d$  : 遅延時間  
 $P_W$  : パルス幅  
 $t_{ri}$  : 入力パルス立ち上がり時間  
 $t_{ri}$  : 入力パルス立ち下がり時間  
 $t_o$  : 出力パルス立ち上がり時間  
 $t_o$  : 出力パルス立ち下がり時間  
 $V_{OL}$  : 出力"L"レベル電圧  
 $V_{OH}$  : 出力"H"レベル電圧



## 2-3 データ・バス・バッファ

データ・バスのバッファにLS245を用いていますが、バッファはバイト・アクセス、ワード・アクセスのどちらにも対応しなければなりません。特にこの回路では、バイトのリード時にも16個のDRAMすべてがデータを出しており、アクティブでない側の8ビットはDRAMからたれ流されたデータとLS245の出力がぶつからないようにしなくてはなりません。

## 2-4 その他の注意点

バンク・レジスタのLS273は本体リセット時に正しく0をもっているように、RESET<sub>0</sub>を使用してクリアしています。また、I/Oデコードには必ずCPUENB<sub>10</sub>もデコードしなくてはなりません。

また、各拡張バスの出力信号は、そのドライブ能力を十分に考えなくてはなりません。1箇所<sup>1</sup>の出力信号から3箇所以上のゲートの入力に接続するのは好ましくなくて2個程度に抑えておくべきです。ただし、本回路ではAB<sub>001</sub>とMRC<sub>0</sub>が3箇所のゲートへ接続され

ています。しかし、これらの信号のファンインを計算するときりぎり拡張バスの能力の中におさまっています。

高密度のDRAMの実装はかなりクリティカルで、特に電源ラインとグラウンドの配線には十分な配慮が必要になります。プリント基板を起こす場合などは、多層基板によってグラウンド・プレーンを回したいところですが、コストの点で大きな難があります。現在市販されている1Mバイト、2MバイトのDRAMボードは、経験的に安定なパターンを得ているようです。

### ③1MバイトDRAMボード用デバイス・ドライバの作成

PC9801シリーズのRAMのメモリ空間は、00000h～9FFFFhまでの640Kバイトしかありませんので、VMの場合で256Kバイト、E/Fの場合で512KバイトまでしかRAMの拡張はできません。

今回使用した1MバイトDRAMボードは、残ったRAMをI/Oポートにあるバンク・レジスタによって、メモリ空間の80000h番地から9FFFFh番地

## 簡単なベンチ・マーク・テスト

80186もV30も、8086よりは命令がいろいろと増加していますが、実際にはほとんど8086のコードで使用されています。そうなると気になるのが速度の問題です。

- ① 8086の代表にPC9801F
- ② 80186の代表にFM16 $\mu$
- ③ 80286の代表にIBM5560

V30は、PC9801VMとPC9801FにV30を乗せたものを用意して速度比較をしてみました。ただし、各システムによってメモリ・タイミングが違ってしまうし、割り込みは禁止して測定しましたが、単純に比較はできません。しかし、相対的にでもなるべく

目安となるように、NOPのループも測定しました。

その結果を表Aに示します。また、その測定プログラムをリストA～リストDに示します。

NOPの速度差を見ると、NOPの基本マシン・サイクルはどのCPUも同じですから、メモリ・システムの速度差がわかります。FやVMの8MHzは7秒台の値となっており、クロック速度が影響しているのがわかります。VXの80286モードでは内部RAMのウェイト数が特に少なくなっており、8MHzのクロックながら高速な動作をしています。

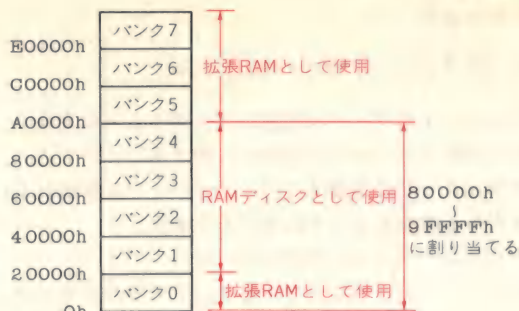
V30の10MHzと80286の8MHzを使用比較すると、実感上は約2倍の速度向上を感じますが、このテストではそれほど速度差はないようです。

VXのスタック動作とサブルーチン・コールの速度

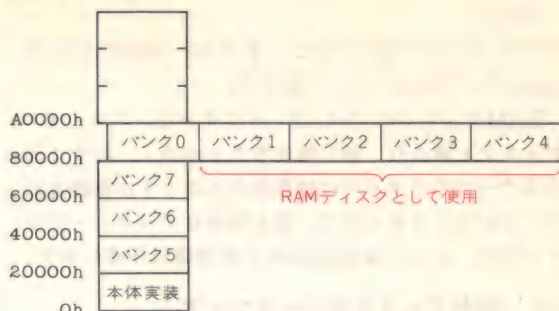
〈リストA〉	-u 100		
ベンチ・マーク	11BE:0100 B91000	MOV	CX,0010
・プログラム、	11BE:0103 89CB	MOV	BX,CX
NOP	11BE:0105 B9FFFF	MOV	CX,FFFF
	11BE:0108 90	NOP	
	11BE:0109 90	NOP	
	11BE:010A 90	NOP	
	11BE:010B 90	NOP	
	11BE:010C 90	NOP	
	11BE:010D 90	NOP	
	11BE:010E 90	NOP	
	11BE:010F 90	NOP	
	11BE:0110 90	NOP	
	11BE:0111 90	NOP	
	11BE:0112 E2F4	LOOP	0108
	11BE:0114 89D9	MOV	CX,BX
	11BE:0116 E2EB	LOOP	0103
	11BE:0118 90	NOP	

〈リストB〉	-u 100		
ベンチ・マーク	11BE:0100 B91000	MOV	CX,0010
・プログラム、	11BE:0103 89CB	MOV	BX,CX
スタック	11BE:0105 B9FFFF	MOV	CX,FFFF
	11BE:0108 50	PUSH	AX
	11BE:0109 50	PUSH	AX
	11BE:010A 50	PUSH	AX
	11BE:010B 50	PUSH	AX
	11BE:010C 50	PUSH	AX
	11BE:010D 58	POP	AX
	11BE:010E 58	POP	AX
	11BE:010F 58	POP	AX
	11BE:0110 58	POP	AX
	11BE:0111 58	POP	AX
	11BE:0112 E2F4	LOOP	0108
	11BE:0114 89D9	MOV	CX,BX
	11BE:0116 E2EB	LOOP	0103
	11BE:0118 90	NOP	

〈図3-1〉 メモリ・バンクの割り当て



(a) 1MバイトRAMボード空間



(b) PC9801のメモリ空間

に128Kバイト単位で切り替えて使用することが可能になっています(図3-1)。このバンクRAMを使用する、MS-DOSのRAMディスクのドライバを作成します。

### 3-1 MS-DOSとデバイス・ドライバ

MS-DOSには、MSDOS.SYSというDOS本体と、各ハードウェアを制御するIO.SYSという標準デバイス・ドライバがあります。さらに新たなハー

ドウェアの制御を行いたい場合には、新たにデバイス・ドライバを作成し、CONFIG.SYSの中に登録しておくことで、容易にデバイス・ドライバの追加が可能となっています。

MS-DOSのデバイスには、

- ① **ブロック型**：フロッピー・ディスク、ハード・ディスク、RAMディスクなど
- ② **キャラクタ型**：コンソール、プリンタ、シリアル回

が他と比較して遅いのは、スタック・ポインタのあったメモリ空間のウェイト数が大きかったためです。VXの内蔵RAMは、0ウェイトの部分と1ウェイトの部分があるようです。内蔵のROMはすべて0ウェイトですが、拡張RAMは4ウェイト、I/O空間は3ウェイトが挿入されています。高速な動作が必要な場合は実行するメモリのアドレスも考えなければならないかもしれません。

V30は、8086に比べて乗除算で速くなっています。PC9801FにV30を差したものは、PC9801VMよりも速い部分があります。これは確かにメモリ・アクセスを伴う処理はFのほうが速くなります。メモリ・リフレッシュがVMになって、75%ほど増加したのが原因と思われます。

〈表A〉 ベンチ・マーク・テスト結果 (単位:秒)

		NOP	スタック	加乗除算	コール
PC9801F(8086)	8 MHz	7.8	17.8	41.0	22.3
PC9801F(V30)	8 MHz	7.0	17.6	12.2	19.9
PC9801VM(V30)	8 MHz	7.0	18.6	12.2	21.0
FM16 <del>8</del> (80186)	8 MHz	6.3	13.7	14.0	15.9
IBM5560(80286)	8 MHz	5.7	9.0	8.8	13.6
PC9801VM(V30)	10MHz	5.6	14.8	10.0	16.7
VX(286)	8 MHz	5.5	9.8	8.2	12.6
VX(V30)	10MHz	5.6	20.7	10.0	20.2

〈リストC〉  
ベンチ・マーク  
・プログラム、  
加乗除算

```

-u 100
11BE:0100 B91000      MOV     CX,0010
11BE:0103 89CB        MOV     BX,CX
11BE:0105 B9FFFF      MOV     CX,FFFF
11BE:0108 89C8        MOV     AX,CX
11BE:010A 01D8        ADD     AX,BX
11BE:010C F7E1        MUL     CX
11BE:010E 31D2        XOR     DX,DX
11BE:0110 F7F1        DIV     CX
11BE:0112 E2F4        LOOP    0108
11BE:0114 89D9        MOV     CX,BX
11BE:0116 E2EB        LOOP    0103
11BE:0118 90          NOP

```

〈リストD〉  
ベンチ・マーク  
・プログラム、  
コール

```

-u 100
11BE:0100 B91000      MOV     CX,0010
11BE:0103 89CB        MOV     BX,CX
11BE:0105 B9FFFF      MOV     CX,FFFF
11BE:0108 E8F500      CALL    0200
11BE:010B E8F200      CALL    0200
11BE:010E E8EF00      CALL    0200
11BE:0111 90          NOP
11BE:0112 E2F4        LOOP    0108
11BE:0114 89D9        MOV     CX,BX
11BE:0116 E2EB        LOOP    0103
11BE:0118 90          NOP

```



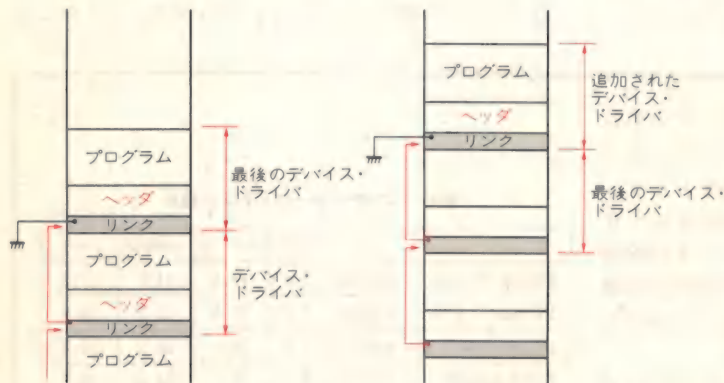
線など  
の大きく二つに分かれています。今回作成するRAMディスクはブロック型です。

RAMディスクのデバイス・ドライバは、フロッピー・ディスクと異なり、差し替えることがなく、必ず1台のみで一つのドライバに複数台のユニットが接続されることがありませんので、最も簡単なデバイス・ドライバです。ここではPC9801E/Fを対象に作成します。

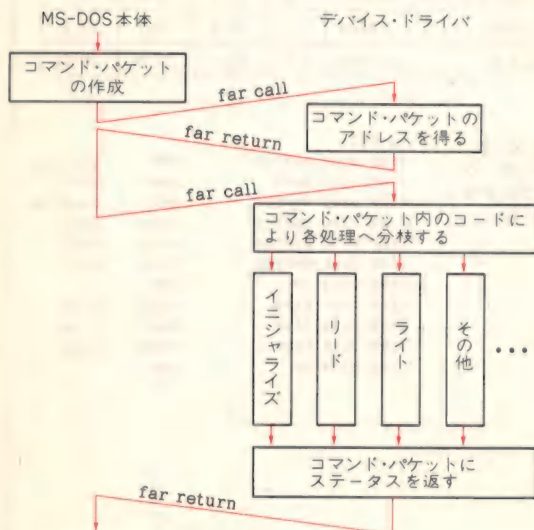
### 3-2 RAMディスクのハードウェア

1MバイトのRAMボードは、128Kバイト単位で8枚のバンクに分けられ、下位よりバンク0からバンク7となります。このうち20000h番地から7FFFh番地の384Kバイトで、バンク5からバンク7までが本体のRAM空間の拡張RAMとなります。80000h番地から9FFFFh番地までの128Kバイトは、通常はバンク0を拡張RAMとして使用し、バンク1からバンク4までの512KバイトをRAMディス

〈図3-2〉 デバイス・ドライバの追加



〈図3-3〉 デバイス・ドライバの処理



クに用います。

### 3-3 デバイス・ドライバのしくみ

デバイス・ドライバは普通のプログラムとは異なり、コードのオリジンが0h番地から始まるバイナリ・コードで、コードの先頭はデバイス・ヘッダと呼ばれるヘッダから始まるという約束があります。

デバイス・ヘッダの先頭には、次のデバイス・ドライバへのリンク・ポインタがあり、すべてのドライバはこのリンクにより管理されています。最後のデバイス・ドライバのリンクは-1となり、リンクの最後を表します。(図3-2)

### 3-4 デバイス・ドライバのエントリ

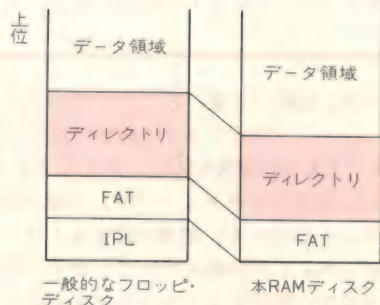
デバイス・ドライバのプログラムのエントリは、

▶ ストラテジ・エントリ

▶ 割り込みエントリ

の2種があり、これらはデバイス・ヘッダ内に書かれ

〈図3-4〉 MS-DOSのファイル構造



〈表3-1〉 BPBの例(本RAMディスクの場合)

dw 512	1セクタの大きさ
db 1	ユニットの領域単位とセクタの関係
dw 0	IPLなどのために使用しないセクタ数
db 1	FATの個数
dw 64	ディレクトリのエントリ数
dw 1024	セクタの総数
db 0FEh	メディアのディスクリプタ
dw 2	FATのセクタ数

ています。

ストラテジ・エントリでは、MS-DOS本体より**コマンド・パケット**という構造体への**ポインタ**を受けとります。コマンド・パケットには、割り込みエントリにより呼ばれたときに行うコマンドとパラメータが格納されています。

割り込みエントリでは、コマンド・パケット内のコマンド・コードにより、12種類の処理へ分岐します。各処理終了後はコマンド・パケットにステータスなどを返し、MS-DOS本体へもどります(図3-3)。

### 3-5 BPB(BIOS Parameter Block)

デバイス・ドライバは、内部にある**BPB**によって**ファイル構造を決定**します。ここでは、1セクタの大きさ、FAT(File Allocation Table)、ディレクトリの大きさなどを自由に決めることができます。RAMディスクからはブートすることはないので、フロッピなどにあるIPL用の領域もなくすることができます(図3-4)。表3-1にBPBの例を示します。

BPBの内容はRAMディスクの大きさにより変更しなければなりません。表3-1の例では1セクタを512バイトとしていますので、512KバイトをRAMディスクに使用できれば、総セクタ数は1024個となります。FATは一つのセクタを1.5バイトを使って管理しますので、1024個のセクタの管理のためには1.5KバイトのFATが必要で、そのためFATのセクタ数は3以上になります。

ここでは、1Mバイトまでの拡張を考えてFATに

6セクタ=3Kバイトを取っています。1セクタの大きさを1024バイトとするとセクタ総数は半分になり、FATに使用するセクタの個数も少なくなりますから、RAMディスク空間の有効利用ができます。デバイス・ドライバを自作する場合にはそのような点に注意して、適当なBPBを作成してください。

### 3-6 プログラムの実際

デバイス・ドライバとして必要なものをまとめると、

- ①**デバイス・ヘッダ**：デバイスの種類や二つのエントリ・アドレス
- ②**BPB**：ファイルの構造を決めるパラメータ・ブロック
- ③**ストラテジ・エントリ**：コマンド・パケットのアドレスを保持する
- ④**割り込みエントリ**：内部で12種の処理に分岐し、実際の入出力を行う

以上の二つのデータ・ブロックと二つのプログラムを用意することが必要です(図3-5)。

表3-2のように、コマンド・コード1はRAMディスクにおいてはメディアの変更はできないので、常に“変更なし”をコマンド・パケットに返せばよく、コマンド・コード2はBPBが一種類のみなので、BPBのポインタを返すだけです。本当に必要なのは、**コマンド・コード0のFATとディレクトリの初期化**、**コマンド・コード4と8のメモリ上でのデータの転送のみ**となります。

初期化は、I/Oのバンク・レジスタに“1”を書き、

トランジスタ技術 別冊  
B5判 304頁 定価1800円

# ソフトマインド<sup>1/</sup>

別冊トランジスタ技術  
トランジスタ技術編集部編

## © on the PC98

CQ出版社

### 目次

- 入門編(1) C言語とはどんな言語／関数ライブラリには何がある？／Cによる簡単なプログラム／オブジェクト・プログラム／アセンブラとの結合
- 入門編(2) DeSmet C (CP/M86, MS-DOS) をPC9801にインプリメントする方法のすべて
- 応用編(1) DOSコール関数ライブラリ
- 応用編(2) Cらしいプログラムを書く
- 応用編(3) ライブラリを作る(リンカ、ライブラリの使い方を含む)
- 応用編(4) ポインタと構造体
- 活用編(1) 通信ソフト(東大 大型計算センターUNIX用)
- 活用編(2) (a) Cソース・プログラムのリスター (tlist)  
(b) PC9801のkey tableの変更
- 活用編(3) コンパクトなスクリーン・エディタ

16ビット・パソコンでは、本格的なプログラムにアセンブラ&C言語を用いる例が増えている。C言語によって、細かい所までプログラムができ、開発時間を短くすることができる。

本誌では、BASICを利用しているユーザが、C言語を学ぶときにぶつかる問題点を、わかりやすく解説している。また、入門書ではフォローできない、パソコンに密着した所、アセンブラと共に使うプログラムも詳しく解説した。

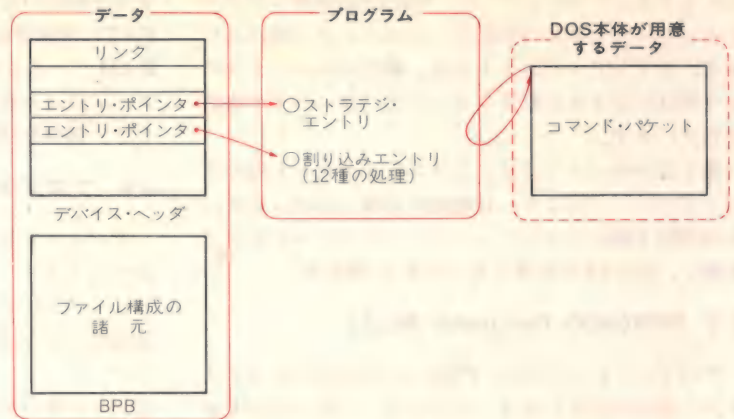
用いたパソコンは、PC9801だが、8086 CPUのマシンの利用者には有益な情報源となるであろう。

解説に用いられたC言語は、ラティス、デスメット、DR、ハイテック、オブティマイズ、マークウィリアムズなどである。



〈図3-5〉

デバイス・ドライバに必要なもの



〈表3-2〉

デバイス・ドライバのコマンド・コード

コマンド・コード	機 能	本 RAM ディスクの処理
0	イニシャライズ	FATとディレクトリの初期化
1	メディアが変更されたかどうかのチェック	メディアは変更できないので簡単
2	BPBの作成	BPBは一種のみなので簡単
3	IOCTL INPUT	RAMディスクには関係なし
4	INPUT (リード)	バンク RAM からのリード
5	NON-DESTRUCTIV INPUT NO WAIT	ブロック・デバイスには関係なし
6	INPUT STATUS	ブロック・デバイスには関係なし
7	INPUT FLUSH	ブロック・デバイスには関係なし
8	OUTPUT (ライト)	バンク RAM へのライト
9	OUTPUT & VERIFY	8と同一処理で簡略化
10	OUTPUT STATUS	ブロック・デバイスには関係なし
11	OUTPUT FLUSH	ブロック・デバイスには関係なし
12	IOCTL OUTPUT	RAM ディスクには関係なし

FATとディレクトリの領域を初期化して、バンク・レジスタに“0”を書き、バンクRAMを元に戻します。このルーチンは最初に一度のみ呼ばれ、デバイス・ドライバ組み込み後は使用されませんので、この領域は開放されてしまいますので、最後尾に書かれています。

リードとライトはセクタ番号からバンク番号の計算とオフセット・アドレスの計算だけ行えば、単純なブロック転送です。ただし、サンプルに挙げたプログラム(リスト3-1)は、80000h番地から9FFFFh番地にあるデータについては正常に動作しません。あくまで参考にしてください。

### 3-7 デバイス・ドライバの登録

RAMディスクのデバイス・ドライバのインプリメントの方法は、

- ①MASMでアセンブルする
- ②LINKする……ここでスタック・セグメントがない

のでワーニングが出るが無視する

③EXE2 BINを用いてバイナリ・イメージを作り、名前をxx.SYSにする。

④CONFIG.SYSの中に  
DEVICE=xx.SYS  
と登録する。

これで、次の立ち上げのときからRAMディスクが使用できます。

日本語入力用のフロントエンド・プロセッサや、拡張RAMを用いたキャッシュ・ディスクなど、有効なデバイス・ドライバが多数発表されています。これらはMS-DOSの拡張性のよさによるものだと思います。

#### 参考文献

- (1) 中村浩次：MS-DOSデバイス・ドライバの作成法，インターフェース1985年2月号，pp. 220～235。
- (2) 標準MS-DOSハンドブック，アスキー出版局。





```

jmp          exit
;
input_:
; es:di = destination pointer address
; cx = number of sector      bpb.start
; dx = sector number        bpb.count

    cmp     cx,0
    jz      exit
    call    setbank
    mov     ds,ax
    xor     si,si
    mov     cx,512/2
    rep     movsw
    mov     al,0
    mov     io_address,al
    out     cx
    pop     dx
    inc     dx
    jmp     input_

always_busy:
    mov     ah,00000001b
    jmp     exitl

;
output:
output_verify:
; es:di = source address pointer
; cx = number of sector
; dx = sector number

    cmp     cx,0
    jz      exit
    push    cx
    mov     ax,es
    mov     ds,ax
    mov     si,di
    call    setbank
    mov     es,ax
    xor     di,di
    cld
    rep     movsw
    mov     cx,512/2
    mov     al,0
    mov     io_address,al
    mov     ai,si
    mov     ax,ds
    mov     es,ax
    pop     cx
    dec     cx
    inc     dx
    jmp     output

;
subrout proc
setbank:
    mov     al,dh
    add     al,1
    out     io_address,al
    mov     ah,0
    mov     al,dl
    mov     al,5

```

```

    shl     ax,cl
    add     ax,08000h
    ret
;
subrout endp
;
exit:
    mov     ah,00000001b      ; done
    exitl:
    lds     bx,cs:[strat_p]
    mov     mov     [bx.status],ax
    es
    pop     ds
    pop     di
    pop     si
    pop     dx
    pop     cx
    pop     bx
    pop     ax
    ret
;
interp endp
;
message db
initialize:
    mov     dx,offset message      ; print message
    mov     ah,9
    int     21h
    mov     al,1
    out     io_address,al
    mov     ax,08000h
    es,ax
    mov     di,di
    xor     di,di
    mov     al,MEDIA_ID
    stosb
    mov     al,0FFh
    mov     al,0
    mov     cx,512*5-3+128*32
    rep     stosb
    mov     al,0
    mov     io_address,al
    out
;
initend:
    lds     bx,cs:[strat_p]
    mov     byte ptr[bx.media],1      ; number of drive
    mov     [bx.count],offset bpb_pointer
    mov     [bx.count+2],cs
    mov     word ptr[bx.trans],offset message
    mov     word ptr[bx.trans+2],cs
    exit
;
ends
;
ramdisk_seg
ramdisk_end

```

## PC9801用A-D/D-A変換ボードの作り方

本章では、データ入力装置として必ず必要になるA-D変換ボードおよびパソコンからアナログ信号を出力するために必要なD-A変換ボードの作り方について詳細に解説します。

最近、測定装置をマイクロコンピュータと接続して計測を行ったり、測定装置自身にマイクロコンピュータが組み込まれることが多くなってきました。これらの理由として、まず第一にデータ収集後の各種の処理がマイクロコンピュータで容易に行うことができることが挙げられます。

例えば、データに対する各種の信号処理(デジタル・フィルタリング、フーリエ変換など)をマイクロコンピュータで行うことができる他、自由にデータを表示することができることや、データの保管がしやすいことなどです。第二にマイクロコンピュータによって自由に測定装置をコントロールすることができ、自動測定が簡単に実現できることです。

最近、マイクロコンピュータの性能が向上したことがこの傾向を促進しています。特に、高速に大量のデータを収集しなければならないような分野にも使用されてきています。PC9801は速度、記憶容量とも比較的に優れており、各種の測定装置と組み合わせて使用するのに適しています。ここでは測定装置の基本ともいえるべき、A-DコンバータをPC9801に接続して使用する方法を紹介します。

また、メカトロニクス・コントロールに必要となるD-A変換ボードの例も示します。

### ①PC9801用A-D変換ボードの製作

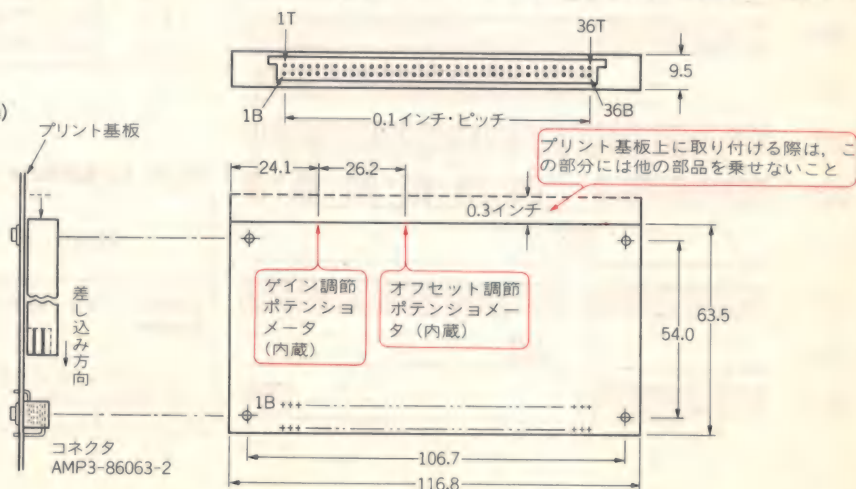
#### 1-1 A-Dコンバータの選定

近年、A-Dコンバータは高速で高精度、高分解能のものが安価で市販されています。これらの中から目的に応じた性能のものを選ぶ必要があります。ここでは、次に挙げる仕様でA-Dコンバータの選択とシステムの設計をしています。

- ① ノイズの影響を低減するために、マイクロコンピュータを測定対象の信号源より離すことと、信号源とデジタル回路をアイソレーションする。
- ② 変換時間：20 $\mu$ s
- ③ 分解能：12ビット
- ④ マイクロコンピュータ側からサンプリング・タイムとチャネル数を自由に変えられること。
- ⑤ サンプリングは外部トリガによって開始されるものとする。

上記の仕様より、A-DコンバータにはMDAS8D(デitel社)を使用しました。これはA-Dコンバータのモジュールで、アナログ・マルチプレクサや、S/H(サンプル&ホールド)、タイミング発生回路を内蔵して

〈図1-1〉  
MDAS8Dの外形寸法(単位：mm)





いる便利なものです。外形寸法を図1-1に示しますが、8チャンネルの差動入力で、解像度は12ビット、スループット率は50kHzと、今回の目的とする仕様に適しています。また、出荷時に較正が行われているので、調整の必要がないのが便利です。仕様の主な内容は表1-1に、内部回路図を図1-2に示します。

8チャンネルのアナログ入力はマルチプレクサMXD807によって1入力を選択され、ボルテージ・フォロワを通り、SHM-LM2によってサンプリングされます。A-DコンバータADC-HZ12BGCによってデジタル信号に変換されます。変換結果のデジタル出力はパラレルおよびシリアル・アウトが用意されています。

またS/H信号や変換開始信号などは、内部のタイミング発生回路によって作り出されます。マルチプレクサのアドレスは外部よりコントロールすることができ

る他、ストローブ信号により自動的に内部アドレス・カウンタをインクリメントさせることもできます。マルチプレクサのチャンネル・アドレスを表1-2に示します。

入力電圧の範囲はピンの接続を変えることによって5種類選択することができます(表1-3)。また、入力電圧範囲を±10Vとすると出力コードは表1-4のようになります。

図1-3にA-Dコンバータのタイミング図を示します。外部よりストローブ信号を受けると内部で6μsのパルスが発生し、その間にマルチプレクサの切り替えとS/Hによるサンプリングが行われ、その後A-Dコンバータが変換を開始します。変換には14μsを要し、合計20μsかかることになります。

A-Dコンバータの出力はパラレルとシリアル・アウトが用意されていますが、シリアル・アウトは14μsの

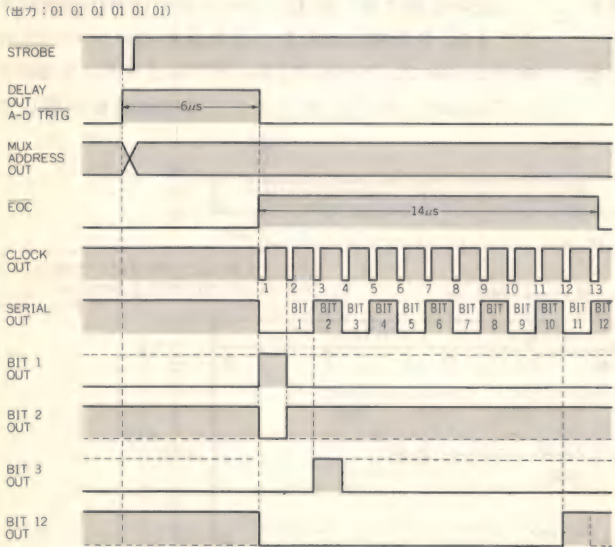
〈表1-1〉 A-DコンバータMDAS8Dの主要スペック

入力チャンネル数	8チャンネル差動入力
入力電圧範囲	0～+5V、0～+10V、±2.5V、±5V、±10V (ピン接続によって選択)
分解能	12ビット(1/4096)
最大誤差 (50kHzサンプリング時)	±0.025%×F.S.
スループット率	50kHz
アキュリゼーション時間	6μsec
A-D変換時間	14μsec
供給電源	+15V±0.5V @ 65mA -15V±0.5V @ 60mA +5V±0.25V @ 200mA

〈表1-2〉 マルチプレクサのチャンネル・アドレス表

ONチャンネル	MUX ENAB.	1	2	4
	(17B)	(22B)	(21B)	(20B)
なし	0	—	—	—
0	1	0	0	0
1	1	1	0	0
2	1	0	1	0
3	1	1	1	0
4	1	0	0	1
5	1	1	0	1
6	1	0	1	1
7	1	1	1	1

〈図1-3〉 MDAS8Dのタイミング図



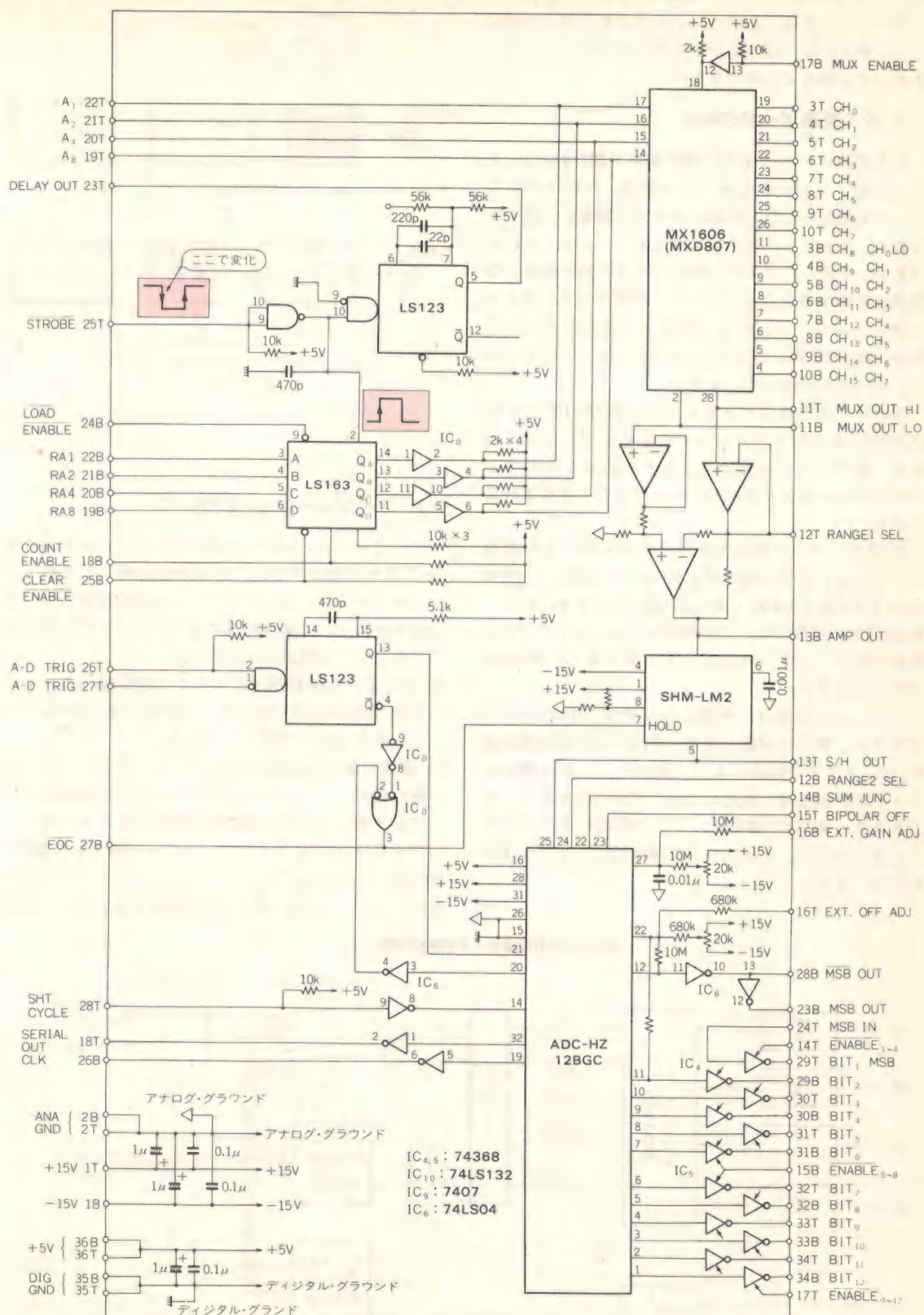
〈表1-3〉 入力電圧範囲設定表

入力電圧範囲	接続すべきピン
0～+5V	12Tと13B, 12Bと13T, 15Tと2T(2B)
0～+10V	12Tと2T(2B), 12Bと13T, 15Tと2T(2B)
±2.5V	12Tと13B, 12Bと13T, 15Tと14B
±5V	12Tと2T(2B), 12Bと13T, 15Tと14B
±10V	12Tと2T(2B), 12Bオープン, 15Tと14B

〈表1-4〉 入力電圧範囲を±10Vとした場合の出力コード表

入力電圧	出力コード	
	MSB bit 1	LSB bit 12
9.9951V (+FS-LSB)	1	1
0.0000V	1	0
⋮	⋮	⋮
-9.9951V (-FS+LSB)	0	0
-10.0000V (-FS)	0	0

〈図1-2〉 MDAS8Dの内部回路とピン接続





間に**MSBより順にクロックと共に出力**されます。シリアル・データは、このクロックの立ち上がりで確定していますから、立ち上がりでラッチすることによってデータを得ることができます。

## 1-2 A-D変換ボードの構成

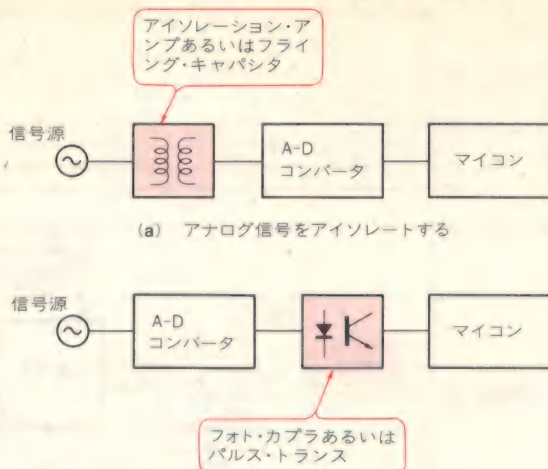
A-D変換ボードの全体の構成概略を図1-4に示します。A-Dコンバータとマイコン側は、外部に対してノイズを発生したり、外部のノイズの影響を低減するために、**フォト・カプラによってアイソレート**されています。さらに、マイコン側のノイズが測定対象の信号源に影響を与えないように、**PC9801を測定対象から離し、A-Dコンバータを測定する信号源の近くに置き、PC9801側のインターフェース・カードとシールド・ケーブルで結んでいます。**

アナログ入力4チャンネルとし、PC9801側よりD<sub>0</sub>、D<sub>1</sub>信号によってコントロールできるようになっています。各アナログ入力はフィルタを通してA-Dコンバータのマルチプレクサのアナログ入力へ接続されています。

A-Dコンバータの出力はシリアル・アウトを使用し、クロック信号と共にフォト・カプラを通して(信号名はDATAとCLK)、PC9801側に送られます。PC9801側の基板では、この信号をシリアル-パラレル変換回路で、パラレル12ビットに組み直し、PC9801が取り込みます。

タイミング回路は、外部よりトリガ(TRG)の入力を受けると動作を開始します。STC(A-D変換開始信号)は、タイマ(8253)によって発生され、その間隔はタイマの初期化時の設定によって自由に変えることができます。RESET信号は、トリガ回路のラッチをクリアする信号で、サンプリング開始前にクリアしなければなりません。

〈図1-5〉 A-Dコンバータのアイソレーション法



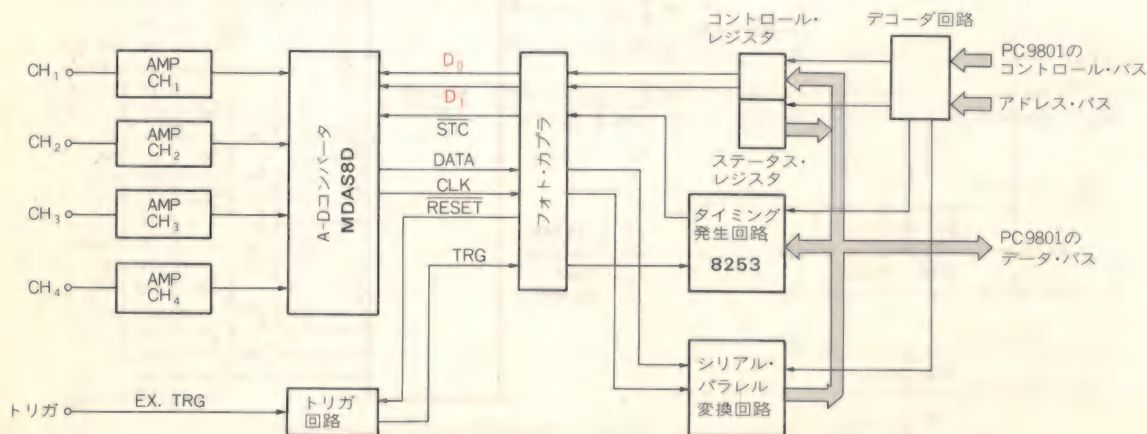
## 1-3 アイソレーションについて

アイソレーションを行う理由は次のようなものです。

- ① アナログ側に飛び込んだ商用周波数ノイズやスパイク・ノイズによってマイコンが誤動作することを防ぐため、工業用計測ではモータやソレノイドなどのノイズが問題となります。
- ② デジタル回路側よりアナログ回路側へのノイズの飛び込みを防ぐため、分解能の高いA-Dコンバータでは特に問題となります。
- ③ 測定対象の信号源の出力インピーダンスが非常に高い場合、マイコンおよびデジタル回路側のノイズが干渉して大きな影響を受けることになります。

アナログ入力をアイソレートする方法として図1-5(a)に示したような**アイソレーション・アンプやフライング・キャパシタを用いる方法**がありますが、やや高

〈図1-4〉 A-D変換ボードの構成概略



〈図1-6〉 フォト・カプラTLP521-4の特性とピン接続図

( $T_a = 25^\circ\text{C}$ )

項	目	記号	測定条件	最小	標準	最大	単位
発 光 側	順電圧	$V_F$	$I_F = 10\text{mA}$	1.0	1.15	1.3	V
	逆電流	$I_R$	$V_R = 5\text{V}$	—	—	10	$\mu\text{A}$
	端子間容量	$C_T$	$V = 0, f = 1\text{MHz}$	—	30	—	pF
受 光 側	コレクタ-エミッタ間降伏電圧	$V_{(BR)CEO}$	$I_C = 0.5\text{mA}$	35	—	—	V
	エミッタ-コレクタ間降伏電圧	$V_{(BR)ECO}$	$I_E = 0.1\text{mA}$	5	—	—	V
	暗電流	$I_D(I_{CEO})$	$I_F = 0, V_{CE} = 24\text{V}$ $T_a = 85^\circ\text{C}$	—	2	50	$\mu\text{A}$
伝 達 特 性	端子間容量	$C_T$	$V_{CE} = 0, f = 1\text{MHz}$	—	10	—	pF
	変換効率	$I_C/I_F$ (注)	$I_F = 5\text{mA}, V_{CE} = 5\text{V}$	50	—	600	%
	コレクタ-エミッタ間飽和電圧	$V_{CE(sat)}$	$I_F = 5\text{mA}, I_C = 1\text{mA}$	—	0.1	0.4	V
	入出力間浮遊容量	$C_S$	$V = 0, f = 1\text{MHz}$	—	0.8	—	pF
	絶縁抵抗	$R_S$	$R.H. = 40 \sim 60\%$ , $V = 1\text{kV DC}$	—	$10^{11}$	—	$\Omega$
	立ち上がり, 立ち下がり	$t_r, t_f$	$V_{CE} = 5\text{V}, R_L = 100\Omega$ $I_C = 2\text{mA}$	—	6	—	$\mu\text{s}$

(注)  $I_C/I_F$  区分A: 50~600, YG: 50~300  
GB: 100~600, Y: 50~150, GR: 100~300, BL: 200~600  
(ただし, YG, Y, GR, BL は TLP521-1 のみ適用)

〈図1-7〉 フォトIC TLP552の特性とピン接続図

( $T_a = 0 \sim 70^\circ\text{C}$ , ただし測定条件中に  $T_a = 25^\circ\text{C}$  が記入のない項目, 標準値(は, すべて  $T_a = 25^\circ\text{C}$  の値))

項	目	記号	測定条件	最小	標準	最大	単位
発 光 側	順電圧	$V_F$	$I_F = 10\text{mA}, T_a = 25^\circ\text{C}$	—	1.65	1.9	V
	順電圧温度係数	$\Delta V_F / \Delta T_a$	$I_F = 10\text{mA}$	—	-2.0	—	$\text{mV}/^\circ\text{C}$
	逆電流	$I_R$	$V_R = 5\text{V}, T_a = 25^\circ\text{C}$	—	—	10	$\mu\text{A}$
受 光 側	端子間容量	$C_T$	$V_F = 0, f = 1\text{MHz}, T_a = 25^\circ\text{C}$	—	45	—	pF
	Hレベル出力電流	$I_{OH}$	$V_{CC} = V_O = 5.5\text{V}$ $I_F = 250\mu\text{A}, V_E = 2.0\text{V}$	—	1	250	$\mu\text{A}$
	Hレベル・インプット電流	$I_{EH}$	$V_{CC} = 5.5\text{V}, V_E = 2.0\text{V}$	—	-1.0	—	$\text{mA}$
伝 達 特 性	Lレベル・インプット電流	$I_{EL}$	$V_{CC} = 5.5\text{V}, V_E = 0.5\text{V}$	—	-1.6	-2.0	$\text{mA}$
	Lレベル出力電圧	$V_{OL}$	$V_{CC} = 5.5\text{V}, I_F = 5\text{mA}$ $V_{EH} = 2.0\text{V}, I_{OL} = 13\text{mA}$	—	0.4	0.6	V
	Hレベル供給電流	$I_{ccH}$	$V_{CC} = 5.5\text{V}, I_F = 0\text{mA}, V_E = 0.5\text{V}$	—	7	15	$\text{mA}$
	Lレベル供給電流	$I_{ccL}$	$V_{CC} = 5.5\text{V}, I_F = 10\text{mA}, V_E = 0.5\text{V}$	—	12	18	$\text{mA}$
	変換効率	$I_O/I_F$	$V_{CC} = 5.0\text{V}, I_F = 5\text{mA}$ $R_L = 100\Omega, T_a = 25^\circ\text{C}$	—	1000	—	%
作 用	入出力間容量	$C_S$	$V = 0, f = 1\text{MHz}, T_a = 25^\circ\text{C}$	—	0.6	—	pF
	絶縁抵抗	$R_S$	$V = 500\text{V}, R.H. = 40 \sim 60\%$ $T_a = 25^\circ\text{C}$	—	$10^{12}$	—	$\Omega$

値になることと, 高い精度を得にくいことがあります。

デジタル入力をアイソレートする方法として図1-5(b)に示すようなフォト・カプラやパルス・トランスを用いる方法があります。このシステムでは, フォト・カプラを用いてアイソレーションしています。今回のように逐次比較型のA-Dコンバータを使う場合は, A-Dコンバータの出力として, パラレル出力は使用せず, シリアル出力を使用してフォト・カプラの数を減らすようにします。

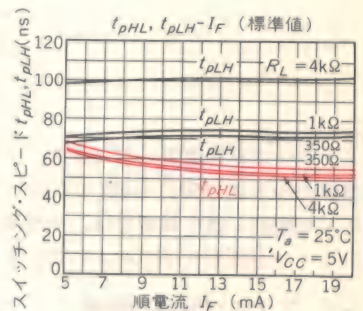
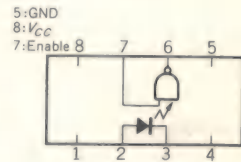
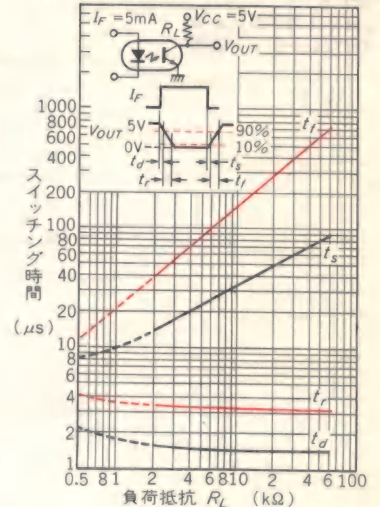
フォト・カプラでアイソレーションされる信号線の

うち, データD<sub>0</sub>, D<sub>1</sub>, RESETの3本の信号は比較的低速なフォト・カプラで十分で, TLP521-4を使用しています。一方, コンバート開始信号のSTC, 出力データDATA, クロックCLK, トリガTRGは1Mビット/sec以上の伝送速度がなくてはなりません。そこで, フォトIC, TLP552を使っています。これは伝搬遅延時間が70 $\mu\text{s}$ と高速なものです。これらのフォト・カプラとフォトICの外形を図1-6, 図1-7に示します。

これらの7本の線は, 外来ノイズの影響を減少させ, かつ外部に対してノイズを放射しないようにシール

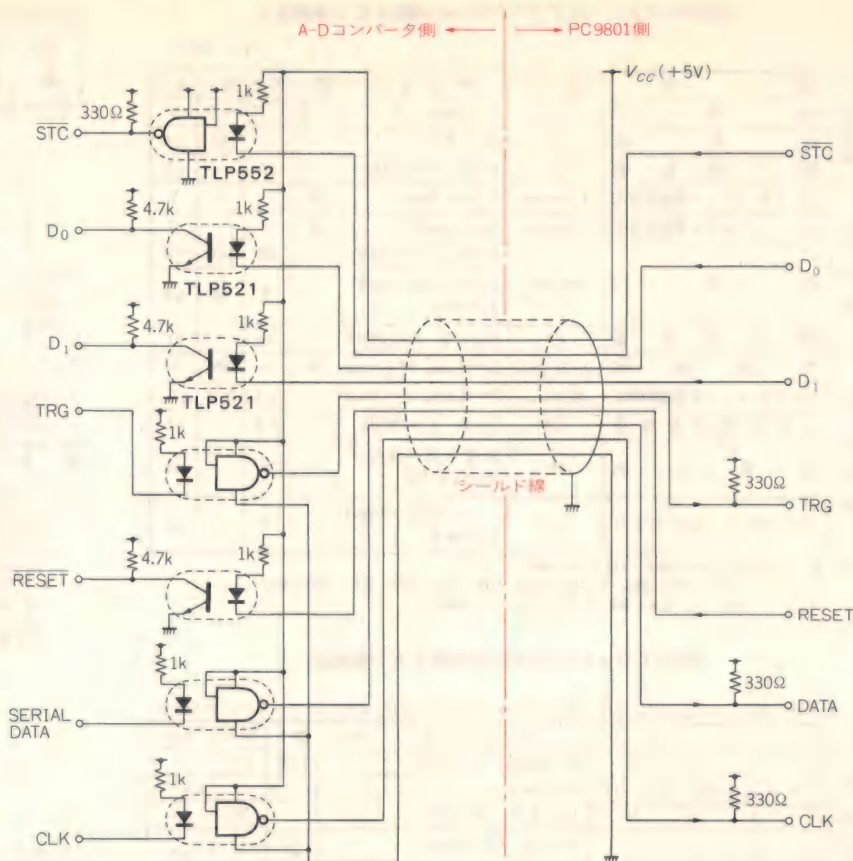


スイッチング時間(飽和動作) (標準時)

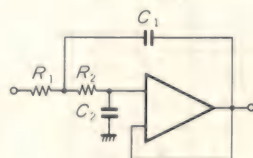




〈図1-8〉  
フォト・カプラによる  
アイソレーション回路



〈図1-9〉  
ローパス・フィルタの設計法



$$f_c = \frac{1}{2\pi\sqrt{R_1 C_1 R_2 C_2}}$$

$$C_1 = \frac{\sqrt{2}(R_1 + R_2)}{4\pi f_c R_2}$$

$$C_2 = \frac{\sqrt{2}}{2\pi f_c (R_1 + R_2)}$$

$R = R_1 = R_2$  で設計すると,

$$f_c = \frac{1}{2\pi R \sqrt{C_1 C_2}}$$

$$C_1 = \frac{\sqrt{2}}{2\pi f_c R} = \frac{0.22508}{f_c R} [\mu]$$

$$C_2 = \frac{\sqrt{2}}{4\pi f_c R} = \frac{0.11254}{f_c R} [\mu]$$

ド・ケーブルを使用しています。フォト・カプラおよびシールド線の部分を図1-8に示します。さらに、ノイズを極度に嫌う環境では、やや高価になりますが光ファイバを使用するのがよいでしょう。

#### 1-4 アナログ回路の構成

S/H回路にサンプル周波数以上の周波数の信号が入った場合、折り返し雑音やビート・ノイズと呼ばれる雑音が発生します。これを生じないようにするためには、信号の中のサンプリング周波数の1/2(ナイキスト周波数)より高い周波数をカットする必要があります。

本システムではこの目的のために、カットオフ特性や、位相特性の点からパワース型フィルタを用いています。図1-9にフィルタ部の回路図とその設計式を示します。本システムは、サンプリング周波数を変えることができますが、最高周波数で使うことを考えて、

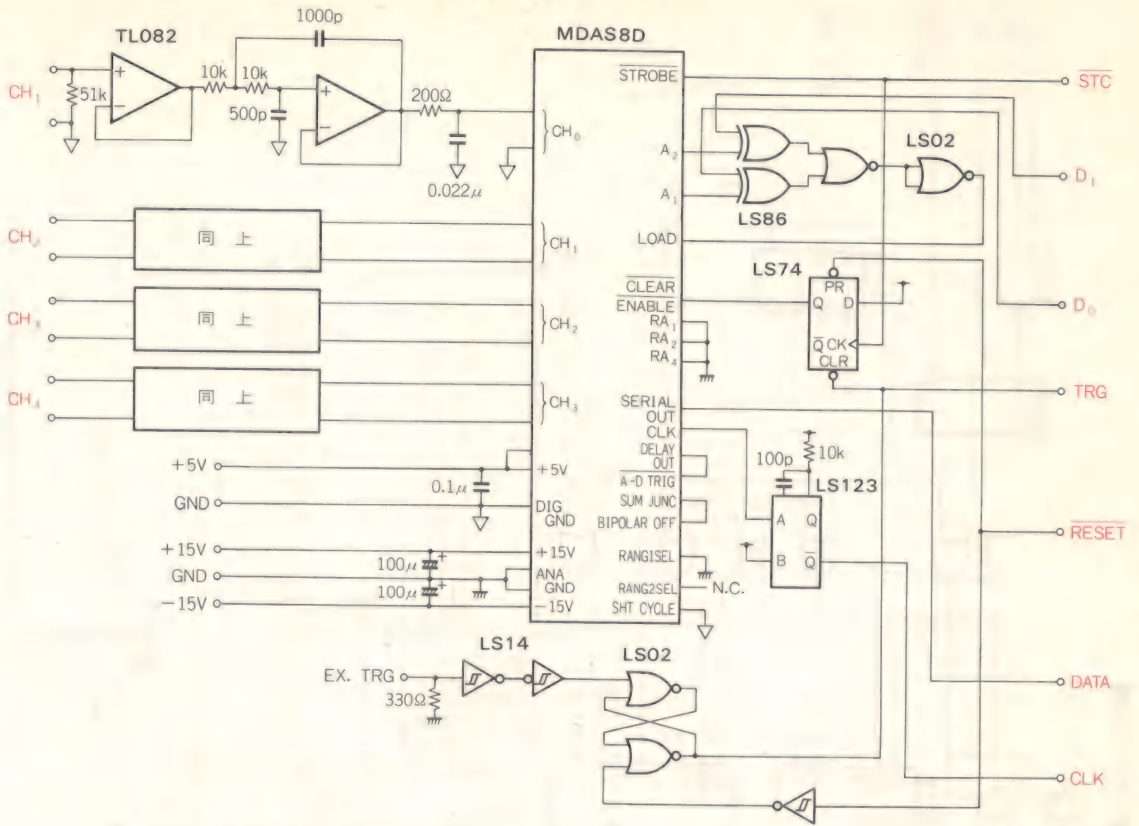
ここではカットオフ周波数を20kHzとして設計しています。

またノイズの点から、アナログ部とデジタル部の電源とグラウンドの両方を独立させるべきでしょう。ありがたいことに今回使用したA-Dコンバータはアナログ部とデジタル部が独立していますから、周辺の回路も独立した給電回路から給電するようにしています。

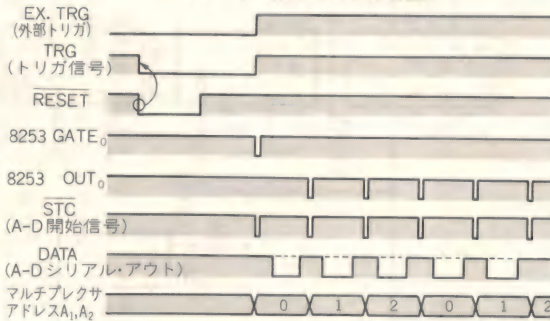
#### 1-5 タイミング発生回路

図1-10にA-Dコンバータ側の回路を、図1-11にPC9801側のインターフェース回路を示します。タイミング回路は、図1-12に示したように外部からのトリガ入力(EX. TRG)によって開始されます。この信号は、“H”でアクティブとなり、ラッチを通してTRG信号としてインターフェース・カードに伝えられます。

〈図1-10〉 A-Dコンバータ側回路



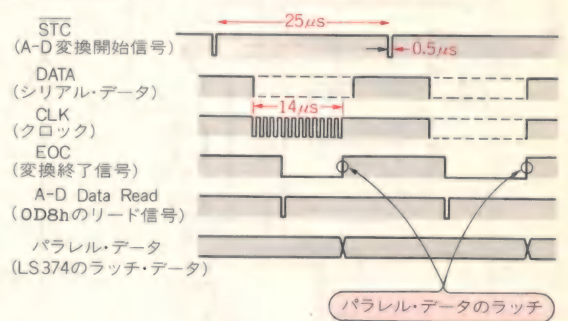
〈図1-12〉 タイミング発生回路のタイミング  
(チャンネル数を3にした場合)



ラッチはトリガの入力前にRESET信号によってクリアされていなければなりません。インターフェース・カードではTRG信号を基に8253に対する開始パルスを作り、GATE<sub>0</sub>に与えます。8253はこのパルスによって動作を開始し、初期設定によって決められた間隔でSTC信号を発生します。

A-DコンバータはこのSTC信号により変換を開始し、その結果をクロックと共にシリアル・データとして出力されます。また、A-Dコンバータのアナログ・マルチプレクサのアドレスは、STC信号により自動的にインクリメントされ、マルチプレクサのアドレス

〈図1-13〉 シリアル-パラレル変換回路のタイミング



がD<sub>0</sub>, D<sub>1</sub>で示されるアドレスと一致するとA-Dコンバータ内のアドレス・レジスタにLOAD命令を出し、値を0に戻します。

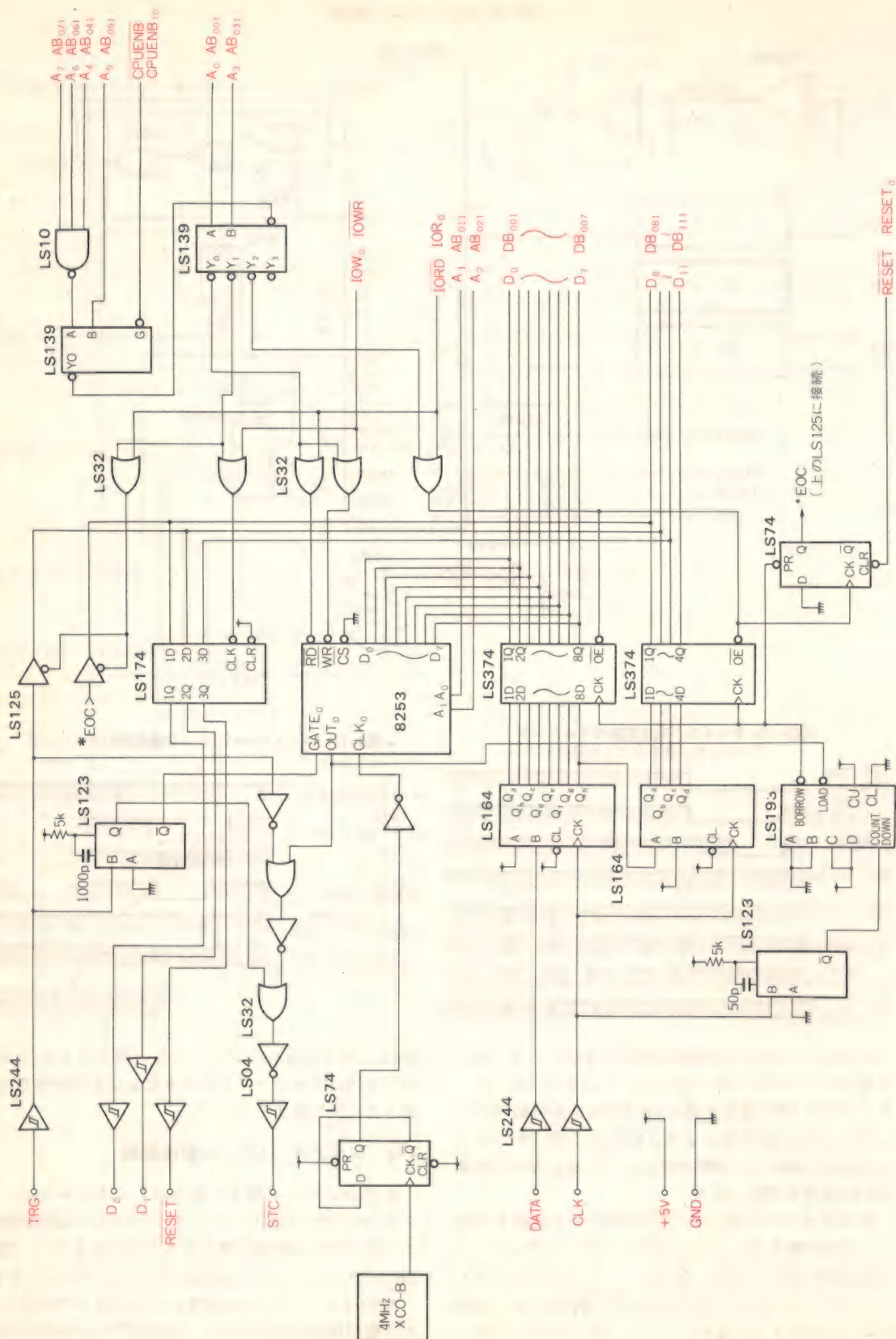
## 1-6 シリアル-パラレル変換回路

A-Dコンバータ側より送られてくるデータはシリアル・データですから、シリアル-パラレル変換回路によってパラレル信号に変えなくてはなりません。図1-13にパラレル-シリアル変換のタイミングを示します。

A-Dコンバータから出力されたクロック信号はパルス幅が100nsと狭いため、LS123でパルス幅を500ns



〈図1-11〉 インターフェース回路



に広くして、PC9801側に送られます。インターフェース・カード側では、送られてきたクロック信号の立ち上がりでシリアル・データをサンプリングして、LS164でパラレル・データへと変換します。変換が終了すると、EOC信号が“H”になり、この立ち上がりでパラレル・データがラッチされると共に、ステータス・レジスタのbit 1が“H”になります。

このフラグをPC9801がポーリングしてA-Dコンバータでの変換が終了したかどうか知るわけです。このEOC信号は、データ・ポートから変換されたパラレル・データを読むことによってクリアされます。

## 1-7 PC9801とのインターフェースとアドレス・デコード

PC9801ではI/OポートのD0HからDFHまでと、E0HからF7Hまでがユーザに対して解放されています。後者の部分は最近RAMディスクなどに使われていますので、今回はD0HからDFHを使うことにしました。

本システムで必要とされるポートは、ステータス・レジスタ用の1バイト、コントロール・レジスタ用の1バイト、A-Dコンバータの出力結果のデータ・レジスタ用の2バイト、8253のコントロール用の4バイトです。各I/Oアドレスを表1-5に示しますが、デコードが完全ではないので、各所にゴーストがあることがわかります。

8086は全部で16ビットのI/Oアドレス空間がありますが、ここでは下位の8ビットのみを使っています。マウス・ボードなどのNEC純正のボードではこのD0HからDFHを使っていることがありますから、この

〈表1-5〉 I/Oアドレス表

	READ	WRITE
D0 H	8253 CH <sub>0</sub>	8253 CH <sub>0</sub>
D1 H	STATUS	CONTROL
D2 H	8253 CH <sub>1</sub>	8253 CH <sub>1</sub>
D3 H	STATUS	CONTROL
D4 H	8253 CH <sub>2</sub>	8253 CH <sub>2</sub>
D5 H	STATUS	CONTROL
D6 H	8253 CTL	8253 CTL
D7 H	STATUS	CONTROL
D8 H	A-D DATA	—
D9 H	—	—
DA H	A-D DATA	—
DB H	—	—
DC H	A-D DATA	—
DD H	—	—
DE H	A-D DATA	—
DF H	—	—

- ▶ D1H, D3H, D5H, D7Hは同じ内容を示しており、デコーダのゴーストのために生じたもの。
- ▶ D8H, DAH, DCH, DEHも同じ

ようなボードを使っている方は、上位8ビットもデコードしたほうがよいでしょう。その回路を図1-14に示しておきます。

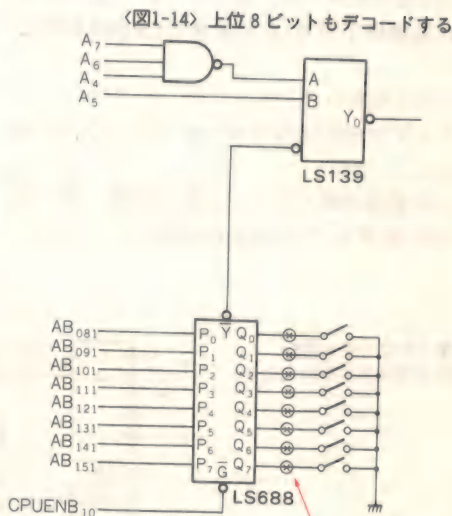
8253は8080系のペリフェラルですから、PC9801に接続する場合は、少し工夫をする必要があります。まず、8253ではRD信号とWR信号より、先にCS信号が“L”に確定していなければなりません。そこで、この回路ではCSを常に“L”にして、RDとWRで8253のバス・バッファをコントロールしています。ですから、8253のRDとWRへは、アドレスのデコードしたものとIOR<sub>0</sub>, IOW<sub>0</sub>の負論理AND(つまりOR)をとったものを入力してやります。

アドレスのデコード時に注意しなければならないことは、CPUENB<sub>10</sub>信号も一緒にデコードしなければならないことです。PC9801ではCPUが動作している場合、このCPUENB<sub>10</sub>信号が“L”になり、リフレッシュ時や8インチ・ドライブのアクセス時にDMAがバスを使っている、CPUが停止している場合“H”になります。

## 1-8 データ・バスとの接続

8086ではI/Oアドレスの偶数番地は、データ・バスの下位8ビットを示し、奇数番地は上位8ビットを示します。また、16ビットを一度にアクセスしたい場合には、偶数番地をアクセスしなければなりません。8253は下位8ビットに接続されており、偶数番地となります。

一方、コントロール・レジスタとステータス・レジスタは上位8ビットに接続されており奇数番地になります。A-Dコンバータの変換データのレジスタは12



3.3kΩで5Vにプルアップ



ビット必要ですから、上位下位両方に接続されており、偶数番地が割り当てられています。ソフトウェアの上からは16ビットと8ビットでは異なってきます。ここについてはあとで示すアセンブラのリストを参考にしてください。

## 1-9 コントロール・レジスタとステータス・レジスタ

ステータス・レジスタは、A-Dコンバータの状態を示すもので、bit 0 は“H”で外部トリガ(TRG)が入力されたかどうかを示し、bit 1 は“H”でA-Dコンバータの変換終了(EOC)を示します。コントロール・レジスタのbit 0 とbit 1 はサンプリングのチャンネル数を決定し、チャンネル数-1を設定します。bit 2 はリセット信号(RESET)で外部トリガのラッチをクリアする信号です。

このbit 2 をサンプリングを開始する前に一度“L”にして、その後“H”に戻すことによって、リセット信号を作る必要があります(図1-15)。ステータス・レジスタとコントロール・レジスタは同じアドレスにありますが、読み出すとステータス・レジスタ、書き込むとコントロール・レジスタになります。

## 1-10 CPUとのインターフェース

今回使用したように、サンプリング時間が15~30  $\mu$ s程度の速度のA-Dコンバータからマイコンでデータを取り込む方法は、一般的に次に挙げるようなものがあります。

- ① DMAによってI/Oから直接メモリ上に高速に転送する。
- ② IOREADY信号を用いて、A-DコンバータからのEOC信号(変換終了信号)がくるまでCPU側を待たせる。
- ③ インタラプトを用いる。
- ④ ソフトウェアでEOC信号をポーリングしてデータを取り込む。

DMAによる転送が最も早く、上記の順番に遅くなります。今回設計した最高20 $\mu$ s程度でしたら、

## 〈図1-15〉 ステータス・レジスタ、コントロール・レジスタ

A-Dコンバータの状態を示す。

bit	7	6	5	4	3	2	1	0
	x	x	x	x	x	x	x	x

EOC  
“1”でデータの用意  
されたことを示す。

TRG  
“1”でトリガがか  
ったことを示す。

(a) ステータス・レジスタ

A-Dコンバータをコントロールする。

bit	7	6	5	4	3	2	1	0
	x	x	x	x	x	x	x	x

RESET  
“L”パルスでトリガ  
のリセットをする。

CH  
サンプリングするチャ  
ネル数-1を指定する。

PC9801ではDMAによる転送を用いる必要はありません。

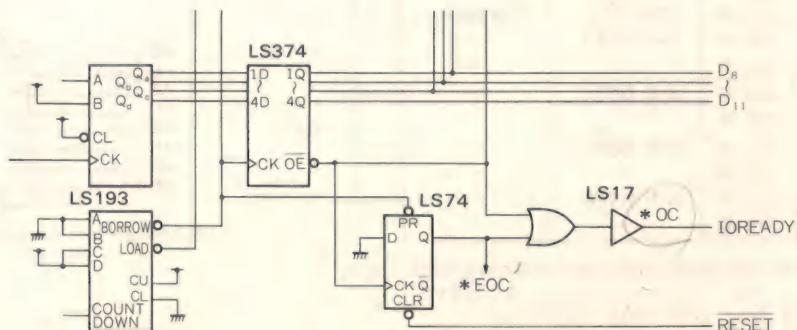
IOREADY信号を用いる方法は、10 $\mu$ s以下でも使うことのできる高速な方法です。CPUからデータの読み出し信号がくると、EOC信号がアクティブになるまで、IOREADYをノンアクティブ“L”にし続け、CPUを待たせます。EOC信号がアクティブになると、IOREADY信号をアクティブ“H”にして、CPUにデータを読み込ませます。

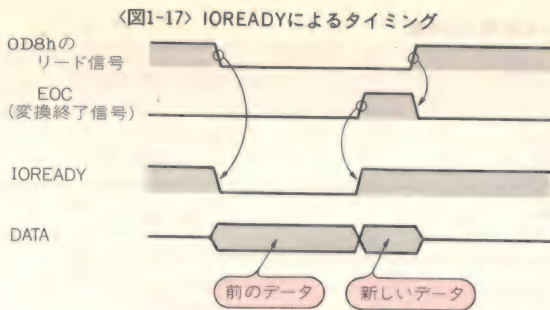
IOREADYを発生する回路は、図1-11の中のEOCの発生回路を図1-16のように変更するだけですみます。この回路とバス間のタイミング図を図1-17に示します。サンプリングのためのプログラムは簡単に書け、リスト1-1のように書くことができます(DIにデータを格納するアドレス、CXにデータ数が入っているものとします)。

しかし問題点があります。サンプリングの間隔が長い場合、DRAMのリフレッシュが正しく行われなくなることです。PC9801では、リフレッシュは約30 $\mu$ s(Vシリーズでは約17 $\mu$ s)に一度タイマから割り込みが入り、DMAによって行われます。ですからこの時間

〈図1-16〉

IOREADYを使ったウェイト回路  
(図1-11のEOCを発生する部分を改変)





〈リスト1-1〉 サンプルング・プログラム

```

smp_loop:
    in      ax,ad_dat      ; sampling
    and     ax,0ffffh
    mov     ds:[di],ax     ; store
    ;
    inc     di
    inc     di
    dec     cx
    jne     smp_loop
    ;

```

を越えてCPUを待たせると、DRAMのリフレッシュが正しく行われなくなり、メモリの内容が壊れる場合が出てきます。

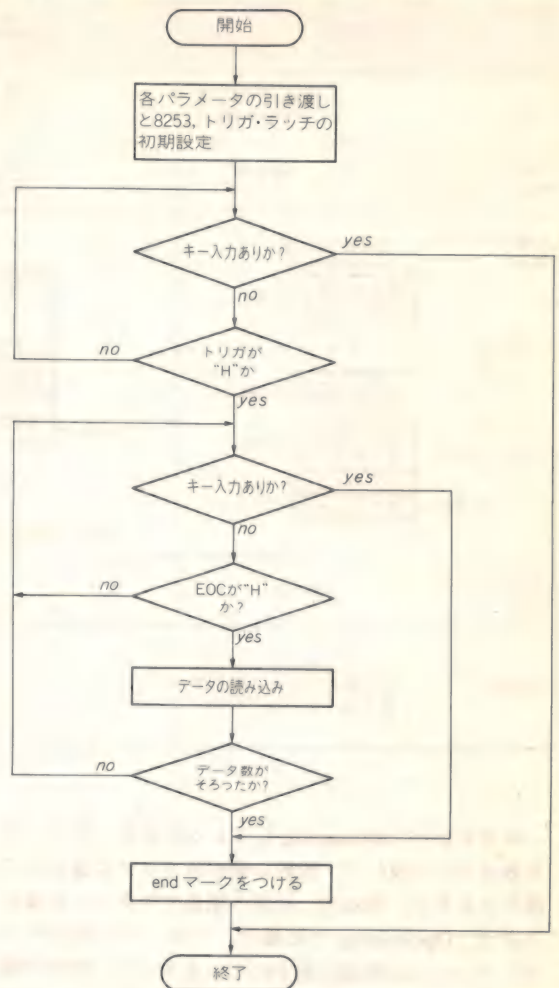
ハードウェア・インタラプトを用いる方法は、②のIOREADYを用いる方法に比べて、やや速率的には遅くなります。これは8086は割り込みがかかると、現在のアドレスをスタックに積み、インタラプト・テーブルを読み、そこに書かれているアドレスにジャンプするという動作をするためで、この分だけ遅くなります。このため、他の動作をしながら、サンプルングする必要のある場合で、もう少し低速のA-Dコンバータには適した方法です。

ここでは④のソフトウェアによってEOC信号をチェックして、データを読み込むポーリングを用いています。8ビット系のZ80などでは20 $\mu$ sのサンプルングはぎりぎりでしたが、PC9801(5 MHzでも)では十分に合います。

図1-18にサンプルングのアルゴリズムを示しますが、これにはキーをチェックする部分があります。外部トリガによって動作するタイプのA-Dコンバータでは、外部からのトリガが何かの原因でこない場合、いつまでもトリガを待って抜け出せなくなります。これを防ぐために、トリガ待ちの状態でキーをチェックして、もし何かのキーが押されていたらこのプログラムから抜け出すようになっています。

同様に、サンプルングの間隔が長い場合、途中で停止したい時、やはりキーを押すことによって抜け出すようになっています。動作が正常に終了した場合や、

〈図1-18〉 ポーリングを用いたアルゴリズム



途中で強制終了させた場合は、どこまでがサンプルングされたデータか示すために、データの最後にこの12ビットA-Dコンバータではありえない0 F F F F Hを8回付けています。

リスト1-2に、図1-18のポーリングを用いたアルゴリズムで書いたアセンブラ言語でのルーチンの例を示します(DIにデータを格納するアドレス、CXにデータ数が入っているものとします)。普通、このようなルーチンを使う場合、高級言語から呼び出して使います。しかし、各言語によって表1-6に示すように引き数の渡し方や、その使い方が異なってきます。そこでMS-FORTRAN, Optimizing-C, N<sub>88</sub>BASIC(MS-DOS)についてその例を示します。

リスト1-3にMS-FORTRANによる例を示します。引き数はアドレス渡しで、変数の入っているアドレスがスタックに積まれて渡されます。注意しなければならないことは、リターン時にスタックを引き数が積まれる以前の状態にもどさなければならないことです。



〈表1-6〉 各言語からの使用上の注意

MS-FORTRAN (Ver 3.0)	Optimizing C	N <sub>86</sub> BASIC (MS-DOS)
引数                      変数のアドレス (アドレス渡し)	変数の内容 (バリュ渡し)	変数のアドレス (アドレス渡し)
引数のフレーム・アドレス      SS : SP を示しているレジスタ          (スタック)	SS : SP (スタック)	DS : BX
引数のフレーム構造 <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">             上位(大きい) ↑ ↓ 下位(小さい)           </div> <div style="border: 1px solid black; padding: 5px;">             ワード(2バイト)              セグメント } 引数1              オフセット              セグメント } 引数n-1              オフセット              セグメント } 引数n              オフセット              セグメント } リターン・アドレス              オフセット              SS:SP →           </div> </div>	ワード(2バイト) <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">             引数nの位              :              引数2の位              引数1の位              オフセット              SS:SP →           </div> <div>             リターン・アドレス              ACQU (引数1, 引数2, ..., 引数n)           </div> </div>	ワード(2バイト) <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">             セグメント } 引数1              オフセット              :              セグメント } 引数n-1              オフセット              セグメント } 引数n              オフセット              DS:BX →           </div> </div>
機械語からの復帰                      RET n	RET	IRET
注意点                      リターン時にはスタックが引数のつまれる前まで戻さなくてはならない。		インターラプトのフラグがディスイネーブルになっているので、イネーブルにしないてはならない。

リスト1-4にOptimizing-Cによる例を示します。引き数はバリュ渡しで、変数の値がスタックに積まれて渡されますが、先ほどとは逆の順番でスタックに積まれます。Optimizing-Cの場合、アセンブラ用のヘッダ・ファイルが各種用意されていますので、宣言が簡単になるほか、@ a bと書いておけば、ビッグ・モデルかスモール・モデルかを判別して、スタックの先頭からパラメータの積んである場所までのオフセットを自動的に付けてくれます。

リスト1-5にN<sub>86</sub>BASIC(MS-DOS)による例を示します。引き数はアドレス渡しですが、引き数用のフレームを決めてあり、DS : BXでその位置を示しています。この例では、配列にデータを入れるのではなく、機械語領域に定義した位置にデータを格納していますので、格納位置を示す引き数はセグメントになっています。

〈リスト1-2〉 ポーリング・ルーチン

```

trg_wat:
    mov     al,es:[si]          ; key scan
    and     al,0ffh
    jne     smp_q
    ;
    in      al,ad_sta           ; triger.wait
    and     al,01
    je      trg_wat
    ;
eoc_wat:
    ; eoc wait
    mov     al,es:[si]          ; key scan
    and     al,0ffh
    jne     smp_end
    ;
    in      al,ad_sta           ; eoc check
    and     al,02
    je      eoc_wat
    ;
    in      ax,ad_dat           ; sampling
    and     ax,0fffh
    mov     ds:[di],ax         ; store
    ;
    inc     di
    inc     di
    dec     cx
    jne     eoc_wat
    ;
smp_end:
    cli
    ;
    mov     cx,8                ; end marker
    mov     ax,0ffffh
mrk_end:
    mov     ds:[di],ax
    inc     di
    inc     di
    loop    mrk_end
    ;

```

〈リスト1-3〉 MS-FORTRANによるA-D変換プログラム

MS-FORTARN

```

C      ARRAY OF SAMPLED DATA
C      INTEGER IDATA(30000)
C      NUMBER OF SAMPLING POINTS
C      INTEGER NUM
C      SAMPLING INTERVAL TIME ( MICRO SECOND )
C      INTEGER IT
C      NUMBER OF CHANNEL
C      INTEGER ICH

```

CALL ACQU( IDATA(1), NUM, IT, ICH )

STOP  
END

```

data      segment 'data'
data      ends
dgroup   group data
code     segment 'code'
assume  cs:code,ds:dgroup,ss:dgroup
;
;
public  acqu
acqu    proc far
ad_dat  equ 0d8h
ad_sta  equ 0d1h
ad_ctl  equ 0d0h
ctc_ctl equ 0d6h
ctc_ch0 equ 0d1h
key_tst equ 0528h
;
push    bp
mov     bp,sp

init:   mov     al,0
out     ad_ctl,al
les     si,[bp+6]
mov     ax,es:[si]
or      al,00000100b
out     ad_ctl,al
;
mov     al,34h
out     ctc_ctl,al
les     si,[bp+10]
mov     ax,es:[si]
add     ax,ax
out     ctc_ch0,al
mov     al,ah
out     ctc_ch0,al
; set 8253 ch0 high byte

; a/d data port
; a/d status
; a/d control register
; 8253 control register
; 8253 channel 0
; key buffer
; reset trigger latch
; es:si = 4th parameter address
; ax = channel number
; set bit2
; set A/D control register
; 8253 initialize
; es:si = 3rd parameter address
; ax = sampling interval
; ax = 2 * ax
; set 8253 ch0 low byte
; set 8253 ch0 high byte

```

```

; les     si,[bp+14]
; mov     cx,es:[si]
; ; es:si = 2nd parameter address
; ; cx = sampling number
; lds     di,[bp+18]
; ; ds:di = 1st parameter address
; ; di = offset address of array
; mov     ax,0
; mov     es,ax
; mov     si,key_tst
; ; es:si = key buffer address
;
; trg_wat:
; mov     al,es:[si]
; and     al,0ffh
; jne     smp_q
; ;
; in      al,ad_sta
; and     al,01
; je      trg_wat
; ;
; eoc_wat:
; mov     al,es:[si]
; and     al,0ffh
; jne     smp_end
; ;
; in      al,ad_sta
; and     al,02
; je      eoc_wat
; ;
; in      ax,ad_dat
; and     ax,0ffh
; mov     ds:[di],ax
; ; store sampling data to ds:[di]
; inc     di
; ; next address
; inc     cx
; ; count down
; jne     eoc_wat
; ; check end
;
; smp_end:
; mov     cx,8
; mov     ax,0fffh
;
; mrk_end:
; mov     ds:[di],ax
; inc     di
; inc     di
; loop    mrk_end
;
; smp_q:
; mov     sp,bp
; pop     bp
; ret     16
; ; 16=(num of param)*4bytes

; acqu
; code
; ends
; end

```



# ＜リスト1-4＞ Optimizing-CによるA-D変換プログラム

Optimizing-C

```
main(
{
    int idata[30000];          /* array of sampled data */
    int num;                   /* number of sampling point */
    int it;                     /* sampling interval time */
    int ich;                     /* number of channel */

```

```
    acqu( idata, num, it, ich );

```

```
    include model.h
    include prologue.h
    public acqu
    acqu proc near
    {

```

```
        ad_dat equ 0d8h          ; a/d data port
        ad_sta equ 0d1h          ; a/d status
        ad_ctl equ 0d0h          ; a/d control register

        ctc_ctl equ 0d6h          ; 8253 control register
        ctc_ch0 equ 0d1h          ; 8253 channel 0

        key_tst equ 0528h         ; key buffer

        push bp
        mov bp,sp
        ;
        di,@ab[bp]               ; di = offset address
        cx,@ab+2[bp]             ; cx = number of sampling

        al,34h
        ctc_ctl,al                ; 8253 initialize
        mov ax,@ab+4[bp]          ; ax = sampling interval
        add ax,ax                 ; ax = 2 * ax
        ctc_ch0,al                ; set 8253 ch0 low byte
        mov al,ah
        ctc_ch0,al                ; set 8253 ch0 high byte

        al,0
        ad_ctl,al                 ; reset trger latch
        mov ax,@ab+8[bp]           ; ax = channel number
        or al,00000100b           ; set bit2
        out ad_ctl,al              ; set A/D control register

        ;
        mov ax,0
        mov es,ax
        mov si,key tst
        ;
    }
}
```

リスト1-2がここに入る

# ＜リスト1-4＞ Optimizing-CによるA-D変換プログラム(つづき)

```
    smp_q: pop bp
        ;
        ret
    ;
endp
include epilogue.h
end
;
```

# ＜リスト1-5＞ N88BASICによるA-D変換プログラム

N88BASIC

```
1000 ' ARRAY OF SAMPLED DATA
1010 ' NUM% NUMBER OF SAMPLING POINTS
1020 ' IT% SAMPLING INTERVAL TIME ( MICRO SECOND )
1030 ' ICH% NUMBER OF CHANNEL
1040 DEF SEG = SEGPTR(2)
1050 MAX%=0
1060 IDATA%=SEGPTR(2)+&H80 ' SEGMENT STORED DATA

2000 CALL MAX( IDATA%, NUM%, IT%, ICH% )

```

cseg	name	acqu	segment	assume	cs:cseg
	org	0			
	ad_dat	equ	0d8h		; a/d data port
	ad_sta	equ	0d1h		; a/d status
	ad_ctl	equ	0d0h		; a/d control register
	ctc_ctl	equ	0d6h		; 8253 control register
	ctc_ch0	equ	0d1h		; 8253 channel 0





## 2 PC9801用D-A変換ボードの製作

ここで紹介するD-A変換ボードは、12ビットのデジタル・データを0～10V、または±5Vのアナログ・データに変換するものです。

前項のA-D変換ボードは、データ収集装置として頻繁に使用されますが、D-A変換はA-D変換に比べると使用頻度はそれほど多くないようです。しかし、パーソナル・コンピュータをメカトロニクスなどに応用するためには重要なものです。

### 2-1 D-A変換ボードの仕様

一般に、メカトロニクス制御では、デジタル・データは12ビット分解能がよく使われます。この理由は、**12ビット分解能はアナログ精度で0.01%に相当**しますが、これ以上の精度を要求しても実現が難しいからです。また、**8ビット分解能はアナログ精度で2%に相当**しますが、これでは少し高精度なコントロールをしたいときには分解能が足りなくなります。

図2-1に、本D-A変換ボードのブロック図を示します。データ・バスは8ビットですから、**下位8ビットと上位4ビットのデータは2回に分けて出力**します。これをラッチに記憶し、12ビット・データとしてD-Aコンバータに出力する構成になっています。

図2-1をみればわかるように2チャンネル構成です。

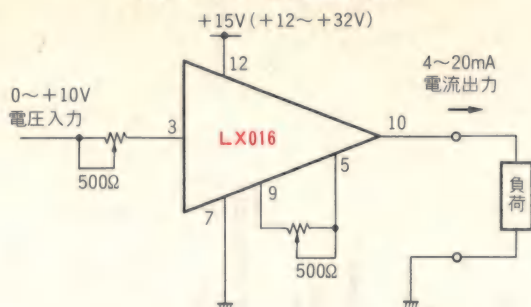
### 2-2 アナログ出力部の構成

図2-2に全回路図を示します。本ボードで使用したD-Aコンバータは、AMD社のAm6012PCで、セトリング・タイムが250ns(typ)です。もう少し高速にしたいときは、セトリング・タイム75ns(typ)のAm6022 PCを使用できます。

この素子は基準電流入力1mAのとき、0～4mAの電流シンク出力です。

本ボードでは、電流-電圧変換にLM308を使用し、

〈図2-3〉 4～20mA電流出力回路



2.5kΩのスパン設定抵抗により10Vのスパンを得ています。±5Vのバイポーラ出力としたいときは、 $TM_3$  ( $TM_6$ )を実装することにより、基準電圧素子REF01(+10V出力)から1/2スケール分の電流2mAを加えます。

コンデンサ $C_1$ 、 $C_4$ はグリッチ除去用ですが、通常はセトリング・タイムを落とすので入れない方がよいでしょう。

本ボードは電圧出力ですが、**計装関係では4～20mAの電流制御も多用されています**。図2-3にCR-BOX(東京無線器材)製のTO8パッケージ入りIC、LX016を使用した回路を示します。

電源に+15Vを使用したとき、0～375Ωの負荷を駆動することができます。

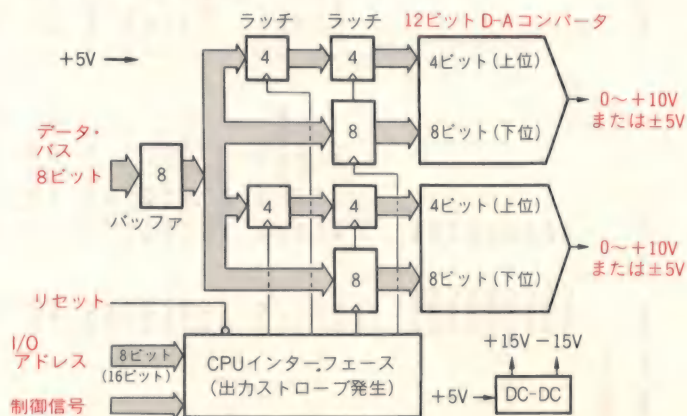
### 2-3 制御部の回路構成

D-Aコンバータは、ラッチ付きのI/Oポートに接続されており、CPUは出力したい電圧値に対応する12ビットのデジタル・コード(バイナリ)をこのポートに書き込むだけです。

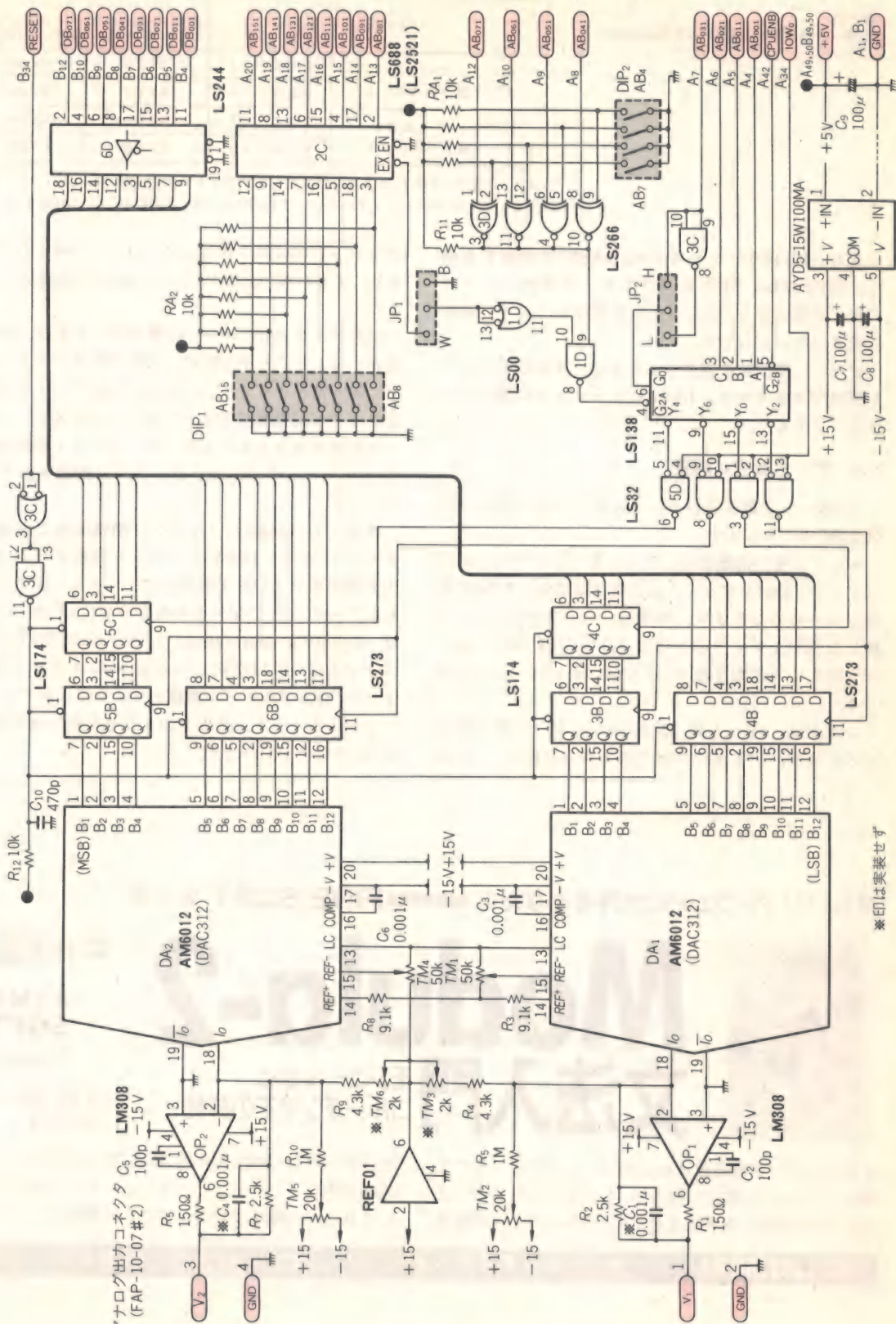
I/Oポートのアドレス割り付けを表2-1に示します。

下位8ビットのうち、上から4ビットまではディップ・スイッチDIP<sub>2</sub>でデコードしますが、次のビットはジャンパ・ポストJP<sub>2</sub>で、最下位3ビット分は74LS138による固定デコードです。

〈図2-1〉 D-A変換ボードのブロック図



〈図2-2〉D-Aボードの全回路図



※印は実装せず



〈表2-1〉  
I/Oポートのアドレス割り付け(Hex表示)

I/O ポート名		I/O アドレス	
		(JP <sub>2</sub> )→L側	(JP <sub>2</sub> )→H側
12ビット/D-A出力1 データ書き込みポート	上位4ビット	XXD0	XXD8
	下位8ビット	XXD2	XXDA
12ビット/D-A出力2 データ書き込みポート	上位4ビット	XXD4	XXDC
	下位8ビット	XXD6	XXDE

(注1) "XX"はDIP<sub>1</sub>による上位8ビット・アドレスの設定値(Hex 2桁)

(注2) ディップ・スイッチの各ビットはONが"0"、OFFが"1"に対応する

なお、PC9801シリーズのN<sub>88</sub>BASICで制御する場合、DIP<sub>2</sub>はHex "D" に限ります。PC9801シリーズでは、内部で多くのI/Oポートを使用しており、ほかに空きがないからです。

ただし、機械語を使用するときは、上位の8ビットも使用できますから、I/Oのロケーションに困ることはありません。

## 2-4 データ形式と調整

I/Oポートに書き込むデータの形式と出力電圧の関係を表2-2に示します。

バイナリを10進数表示したものをデジタル・マグニチュードDMとすると、出力電圧はこれに2.5mVを乗じたものになります。本来は、スパン10Vを2<sup>12</sup>で除した値が1ディジット分(2.4414mV)なのですが、少し拡大して区切りのよい2.5mV/ディジットとしました。

この表は、0V～+10.2375Vのユニポーラ出力時のものですが、バイポーラ出力のときは1/2スケールの

オフセットが印加されるため、DM=2048で出力電圧0V、すなわち-5.120V～+5.1175V出力範囲となります。

出力データをI/Oポートに書き込むときは、必ず上位4ビット・データを先に、次に下位8ビット・データの順で書き込みます。これは、上位データを受け取るラッチが二重構成となっており、下位8ビットのデータが書き込まれるのを待って、12ビット分が同時にD-Aコンバータに印加されるように配慮されているからです。

本ボードの調整は、OPアンプ増幅回路と同様に、スケール(ゲイン)調整、オフセット調整を行います。出力電圧をデジタル電圧計でモニタしながら、表2-2にしたがって、DM=0のとき0V(バイポーラならば-5.120V)、DM=4095のとき+10.2375V(バイポーラならば+5.1175V)となるように、オフセット調整トリマ、およびスケール調整トリマを回します。

チャンネル1とチャンネル2のトリマの割り当ては表2-3のようになっています。

新しいソフトウェアの世界をあなたに — FINE SOFT 第2弾



# Modula-2

## 文法入門

モジュールから  
コルーチンまでの詳細

中村和郎 著

**FINE  
SOFT**  
series

A5判, 168ページ  
1,300円

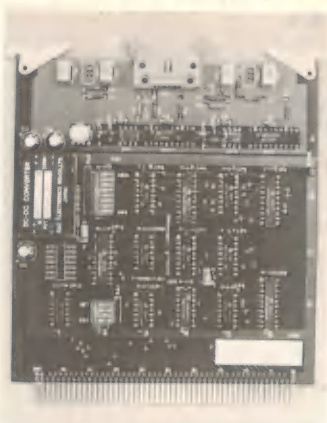
すぐれたデータ構造をPascalから、モジュールと分離コンパイルをModulaから、それぞれうけつぎ、さらに「並行処理」を実現するため「コルーチン」の概念

を採り入れたModula-2。本書は、そのModula-2の基本的なことから、モジュール、コルーチン、そして低レベル機能までを、やさしく解説しています。

**CQ出版社**

〒170 東京都豊島区巣鴨1-14-2 ☎03(947)6311 振替東京0-10665

〈写真2-1〉 完成したD-Aボードの概観



〈表2-2〉 データ形式と出力電圧

(上位4ビット・ポート)								(下位8ビット・ポート)								出力	
ビット7	6	5	4	3	2	1	0	ビット7	6	5	4	3	2	1	0	DM	VO
#	#	#	#	1	1	1	1	1	1	1	1	1	1	1	1	4.095	10.2375 V
}								}								}	}
#	#	#	#	1	0	0	0	0	0	0	0	0	0	0	0	2.048	5.1200 V
}								}								}	}
#	#	#	#	0	0	0	0	0	0	0	0	0	0	0	1	1	0.0025 V
#	#	#	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0000 V

(注1) “#”は“1”、“0”、いずれでもよい。

(注2) DMの算出: 上位4ビット・ポートに書き込まれたデータの10進表示をDH、  
下位8ビット・ポートに書き込まれたデータの10進表示をDLとすると、  
 $DM = (256 * DH) + DL$  となる

## 2-5 プログラミングについて

出力したい電圧に相当するデータをI/Oポートに書き込むだけです。例えば、アナログ出力1に+10.2375Vを得たいときの操作を、N<sub>88</sub>BASICで書くと、

```
OUT &HDO, &HOF
```

```
OUT &HD2, &HFF
```

となります。

リスト2-1にPC9801シリーズの拡張I/Oスロットに挿入し、N<sub>88</sub>BASICで操作するプログラムを示します。  
また、リスト2-2に同じく機械語で操作するプログラム例を示します。

〈表2-3〉 調整トリマの割り当て

	スケール調整	オフセット調整	
		ユニポーラ	バイポーラ
D-A出力1	TM <sub>1</sub>	TM <sub>2</sub>	TM <sub>3</sub>
D-A出力2	TM <sub>4</sub>	TM <sub>5</sub>	TM <sub>6</sub>

〈リスト2-1〉 BASICによるプログラム例

```

100 ***** DA コンバータ テスト プログラム (ヘッダ) for PC-9801series *****
120
130 WIDTH 40,20
140
150 JP2$="L" ' I/O アドレス AB031 ノ セッティ (L,H;シヤンハ-ホ*ストJP-2)
160 MODE$="UNI" ' ユニポーラ(UNI) 、 ハイポーラ(BIP) ノ セッティ
170 ' ...[ DIP-2 ハ "D" , JP-1 ハ B( BASICモード) ニ シテク*サイ ]...
180
190 IF JP2$="L" THEN A$="0" ELSE IF JP2$="H" THEN A$="8" ELSE PRINT"see150":END
200 PORT=VAL("&hD"+A$) ' I/O アドレス カイ 8ビット ノ セット
210
220 ' D/A ホ-ート 1 ノ セッティ
230 INPUT port1=","V1$ ' D/A output(port1) ノ ニュウリョク
240 IF MODE$="UNI" THEN IF V1$<0 OR 10.2375$<V1$ THEN 230
250 IF MODE$="BIP" THEN IF V1$<-5.12 OR 5.1175<V1$ THEN 230
260 I=INT(V1$/.0025) : IF MODE$="UNI" THEN DM1=I ELSE DM1=I+2048
270 PRINT " <true PORT1 =" ;I*.0025;">"
280
290 ' D/A ホ-ート 2 ノ セッティ
300 INPUT port2=","V2$ ' D/A output(port2) ノ ニュウリョク
310 IF MODE$="UNI" THEN IF V2$<0 OR 10.2375$<V2$ THEN 300
320 IF MODE$="BIP" THEN IF V2$<-5.12 OR 5.1175<V2$ THEN 300
330 I=INT(V2$/.0025) : IF MODE$="UNI" THEN DM2=I ELSE DM2=I+2048
340 PRINT " <true PORT2 =" ;I*.0025;">"
350
360 DM1H=INT(DM1/256) : DM1L=DM1-256*DM1H ' 1
370 DM2H=INT(DM2/256) : DM2L=DM2-256*DM2H ' 1 デシマル マクニチュ-ト ノ セット
380
390 OUT PORT+0,DM1H ' 1
400 OUT PORT+2,DM1L ' 1 D/A OUTPUT 1 ハ シュツリョク
410 OUT PORT+4,DM2H ' 1
420 OUT PORT+6,DM2L ' 1 D/A OUTPUT 2 ハ シュツリョク
430
440 PRINT : GOTO 230 ' クリカエシ

```



〈リスト2-2〉 機械語によるプログラム例

```

100 '*** DA コンパクタ テスト プログラム (マシンゴ) for PC-9801series ***
120 '
130 CLEAR ,&H1D00 ' RAMエリア,マシンゴ セントウ ノ センゲン
140 WIDTH 40,20
150 '----- ハ ラメータ ノ セツタイ -----
160 UPBYT$="FF" ' I/O アドレス シ ヨウイ 1ハイト ノ セツタイ (&H00-FF;DIP-1)
170 MODE$="UNI" ' ユニホーラ(UNI) 、 ハ イホーラ(BIP) ノ セツタイ
180 ' ..[ DIP-2 ハ "D",JP-1 ハ W(ワート モート),JP-2 ハ L ニ シテクタ サイ ]..
190 '
200 DEF SEG=&H1D00
210 RESTORE 440:FOR I=0 TO &H40:READ X$:POKE I,VAL("&h"+X$):NEXT I
220 OUTPUT=0
230 DM0%=VAL("&h"+UPBYT$) ' キカイゴ サブルーチン トノ リンク
240 '
250 ' D/A ホート 1 ノ セツタイ
260 INPUT " port1 =",V1$ ' D/A output(port1) ノ ニュウリョク
270 IF MODE$="UNI" THEN IF V1$<0 OR 10.2375#<V1$ THEN 260
280 IF MODE$="BIP" THEN IF V1$<-5.12 OR 5.1175<V1$ THEN 260
290 I=INT(V1$/.0025) : IF MODE$="UNI" THEN DM1%=I ELSE DM1%=I+2048
300 PRINT " <true PORT1 =" ;I*.0025;">"
310 '
320 ' D/A ホート 2 ノ セツタイ
330 INPUT " port2 =",V2$ ' D/A output(port2) ノ ニュウリョク
340 IF MODE$="UNI" THEN IF V2$<0 OR 10.2375#<V2$ THEN 330
350 IF MODE$="BIP" THEN IF V2$<-5.12 OR 5.1175<V2$ THEN 330
360 I=INT(V2$/.0025) : IF MODE$="UNI" THEN DM2%=I ELSE DM2%=I+2048
370 PRINT " <true PORT2 =" ;I*.0025;">"
380 '
390 CALL OUTPUT(DM0%,DM1%,DM2%)
400 '
410 PRINT : GOTO 230 ' クリカエシ
420 '
430 '----- machine code subroutine -----
440 DATA 06 : '0000 PUSH ES
450 DATA 56 : '0001 PUSH SI
460 DATA 50 : '0002 PUSH AX
470 DATA 53 : '0003 PUSH BX
480 DATA 51 : '0004 PUSH CX
490 DATA 52 : '0005 PUSH DX
500 '
510 DATA 8B,4F,0A : '0006 MOV CX,10[BX]
520 DATA 8E,C1 : '0009 MOV ES,CX
530 DATA 8B,77,08 : '000B MOV SI,8[BX]
540 DATA 26,8A,34 : '000E MOV DH,ES:[SI]
550 '
560 DATA 8B,4F,06 : '0011 MOV CX,6[BX]
570 DATA 8E,C1 : '0014 MOV ES,CX
580 DATA 8B,77,04 : '0016 MOV SI,4[BX]
590 DATA 26,8A,44,01 : '0019 MOV AL,ES:1[SI]
600 DATA B2,D0 : '001D MOV DL,D0
610 DATA EE : '001F OUT DX,AL
620 DATA 26,8A,04 : '0020 MOV AL,ES:[SI]
630 DATA B2,D2 : '0023 MOV DL,D2
640 DATA EE : '0025 OUT DX,AL
650 '
660 DATA 8B,4F,02 : '0026 MOV CX,2[BX]
670 DATA 8E,C1 : '0029 MOV ES,CX
680 DATA 8B,37 : '002B MOV SI,0[BX]
690 DATA 26,8A,44,01 : '002D MOV AL,ES:1[SI]
700 DATA B2,D4 : '0031 MOV DL,D4
710 DATA EE : '0033 OUT DX,AL
720 DATA 26,8A,04 : '0034 MOV AL,ES:[SI]
730 DATA B2,D6 : '0037 MOV DL,D6
740 DATA EE : '0039 OUT DX,AL
750 '
760 DATA 5A : '003A POP DX
770 DATA 59 : '003B POP CX
780 DATA 5B : '003C POP BX
790 DATA 58 : '003D POP AX
800 DATA 5E : '003E POP SI
810 DATA 07 : '003F POP ES
820 DATA CF : '0040 IRET
830 '-----

```

## PC9801用メカトロニクス・インターフェース・ボードの作り方

本章では、メカトロニクスにパソコンを応用するときに必要なアイソレート入出力ボード、直流SSR出力ボード、アップ／ダウン・カウンタ・ボード、ステッピング・モータ制御ボードの作り方について詳細に解説します。

### ①アイソレート入出力ボードの製作

パーソナル・コンピュータと外部周辺機器とのインターフェースにおいて、データの入出力には**ON/OFF入出力**が基本的に必要です。

ここでは、メカトロニクス制御に利用できるアイソレート入出力ボードの製作例を示します。

最近では、パーソナル・コンピュータを使用して工場での組み立てラインを構成したり、制御盤などの制御もパーソナル・コンピュータで行うことが多くなっています。しかし、これらの**FA分野でパーソナル・コンピュータを利用するには、対ノイズ性を考慮せずに組み込むことはできません**。コンピュータからの入出力(リレー、電磁弁、リミット・スイッチ)の制御を行うことを考えると、誘導ノイズや高周波ノイズ、グラウンド・レベルの変化などの影響により、コンピュータが誤動作する原因となります。

このようなノイズの対策としては**フォト・カプラによる絶縁(アイソレート)**が有効で、制御機器などとインターフェースする場合に多く使用される技術です。

ここで紹介するフォト・カプラを使用した8チャンネル入力、16チャンネル出力用アイソレート・ボードは、

PC9801シリーズのパーソナル・コンピュータの拡張I/Oスロットに挿入することで、外部とのインターフェースが容易に行えます。

#### 1-1 アイソレート入出力ボードの仕様と構成

図1-1に本ボードのブロック図を、図1-2に本ボードの全回路図を示します。**出力部は16チャンネル分あり、ビット単位の出力ができます**。最大定格は、電圧30V、電流0.75A、電力1Wです。

入力部は8チャンネル分あり、出力と同様に**ビット単位の入力が可能です**。入出力共に最大伝達速度は200 $\mu$ sとなっています。

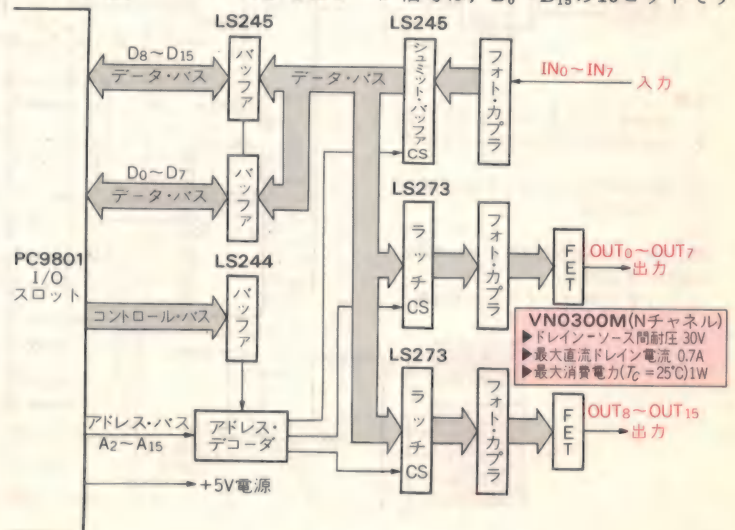
また、ボードの先頭アドレスの設定は、2個のディップ・スイッチによって任意に設定できます。偶数と奇数アドレスの選択もでき、PC9801側の狭いI/O空間を効率良く使用できます。

PC9801のバス側は、シュミット・バッファ(LS245)を用い、PC9801バス上の負荷を少なく、ノイズ・マージンを強くするなどの配慮をしています。

I/Oアドレス・デコード回路では、 $A_2 \sim A_{15}$ の14アドレスをディップ・スイッチ( $SW_1, SW_2$ )により、先頭アドレスを任意に設定可能です。

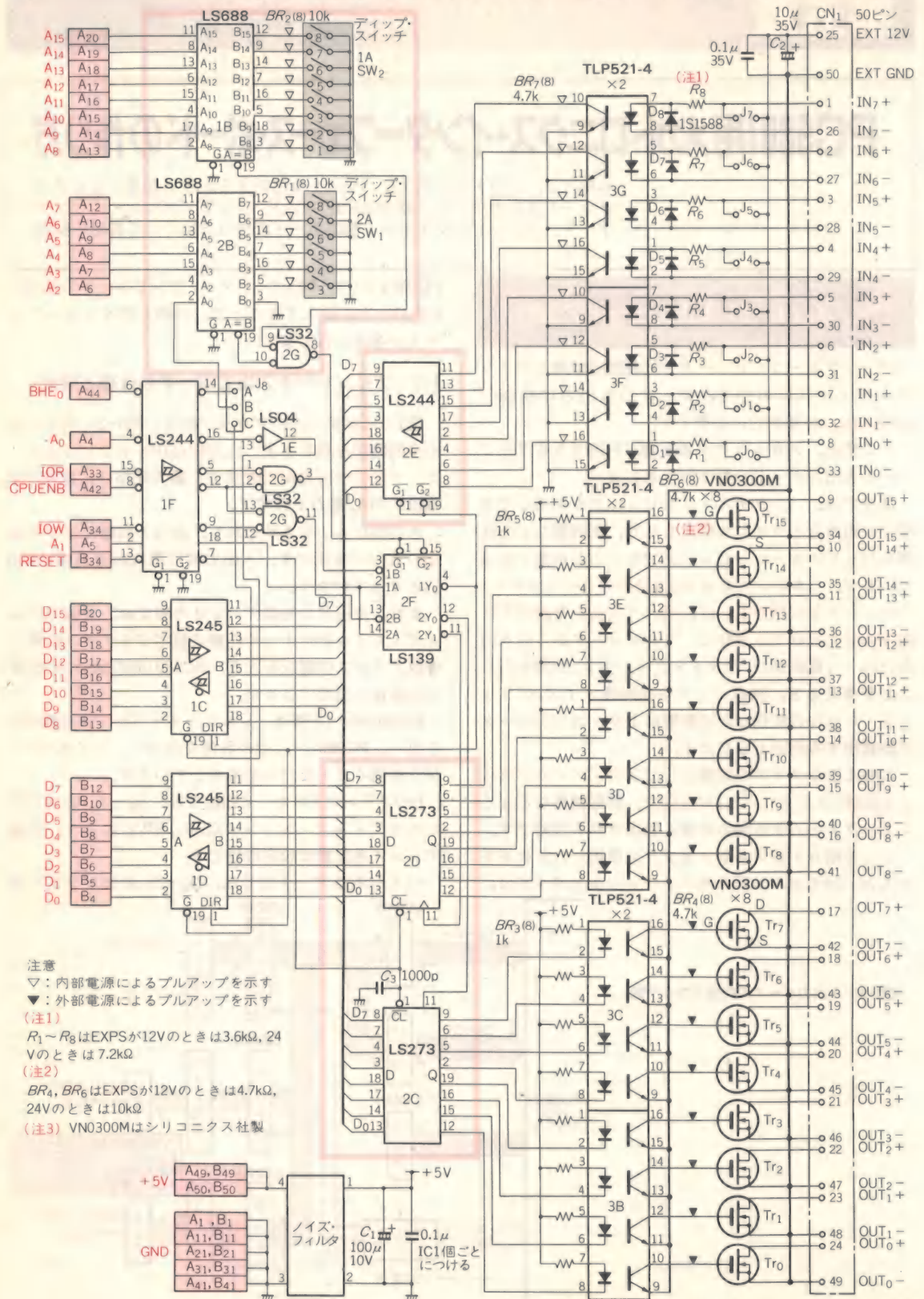
バスからのデータ信号は、 $D_0 \sim D_{15}$ の16ビットです

〈図1-1〉 入出力ボードの内部ブロック図





〈図1-2〉 入力8チャンネル、出力16チャンネル・アイソレート入出力ボードの回路図



が、本ボードでは下位と上位を切り替えることにより、いずれかの8ビットを使用します。

I/Oの入力部は、フォト・カプラを通り、シュミット・バッファ(LS244)に入力されます。出力部は、8ビットのラッチ(LS273)を2個使用し、下位1アドレスで切り替え、8ビット×2ポートを構成しています。

ラッチからの出力は、フォト・カプラを通り、**パワーマOS FETによるバッファ**に入力されます。パワーマOS FETは、入力インピーダンスが高く、出力のオン抵抗が小さいという特徴があるため、フォト・カプラの出力負荷を小さくでき、スイッチング特性を向上できます。

また、出力インピーダンスも非常に小さいために、負荷損失が少なく、TTLレベルの出力も可能であり、バッファに適しています。インターフェースのコネクタ部は、DIN型50ピン・コネクタを用いて外部と接続できるようになっています。

## 1-2 本ボードの操作について

操作の例として、PC9801のBASICのI/O命令による方法を説明します。

このボードのI/O先頭アドレスは、SW<sub>1</sub>下位(A<sub>2</sub>~A<sub>7</sub>)、SW<sub>2</sub>上位(A<sub>8</sub>~A<sub>15</sub>)によりアドレスを設定します。**I/Oアドレスの設定で機械語を使用しない場合は、上位アドレスは“00”にセットし、下位アドレスのみで設定を行います。**この例ではD0番地を先頭アドレスとし、表1-1にアドレスと機能の対応表を示します。

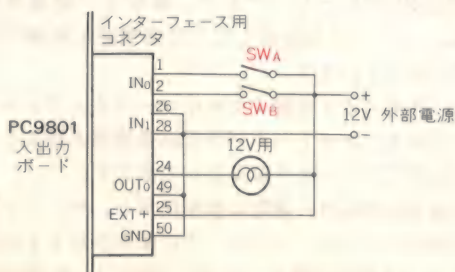
図1-3の回路を用いて説明します。この例では、入力IN<sub>0</sub>にSW<sub>A</sub>、IN<sub>1</sub>にSW<sub>B</sub>が接続され、出力ではOUT<sub>0</sub>にランプが接続されています。プログラムにより、入力のSW<sub>A,B</sub>の状態を見るには、

```
A=INP(&HDO)
PRINT A
```

〈表1-1〉アドレス対応表

I/Oアドレス	リード	ライト
D0	IN <sub>0</sub> ~IN <sub>7</sub> リード	OUT <sub>0</sub> ~OUT <sub>7</sub> ライト
D2	—	OUT <sub>8</sub> ~OUT <sub>15</sub> ライト

〈図1-3〉入力にスイッチ、出力にランプをつないで動作確認



〈リスト1-1〉動作のチェック・プログラム

```
10 *****
20 * AB9B-10A 16CH-IN 8CH-OUT SAMPLE PROGRAM *
30 *
40 *****
50 *
60 *----- スタート -----
70 *
80 *START
90 CLS:J=0:I=0
100 *
110 *
120 *----- 8ビット・データ入力 -----
130 *
140 *IN
150 IN=INP(&HDO)
160 LOCATE 9,5:PRINT "こウリクツ" "A=";IN
170 *
180 *----- 比較&出力ON -----
190 *
200 IF IN<>1 THEN 250
210 OUT &HDO,1
220 LOCATE 9,5:PRINT "ランプ" ON."
230 FOR J=0 TO 1000:NEXT J
240 GOTO *START
250 *
260 *----- 比較&出力ON/OFF -----
270 *
280 IF IN<>2 THEN 380
290 LOCATE 9,5:PRINT "ランプ" ON/OFF."
300 *LOOP
310 I=I+1
320 OUT &HDO,I
330 FOR J=0 TO 1000:NEXT J
340 OUT &HDO,0
350 FOR J=0 TO 1000:NEXT J
360 IF I<>5 THEN *LOOP
370 GOTO *START
380 *
390 *----- 比較&出力OFF -----
400 *
410 OUT &HDO,&H0
420 LOCATE 9,5:PRINT "ランプ" OFF."
430 FOR J=0 TO 1000:NEXT J
440 *
450 END
```

をキーボードから入力すると、画面上に10進数で入力データが表示されます。SW<sub>A,B</sub>がOFFの場合には“0”が表示され、SW<sub>A</sub>のみONの場合には“1”、SW<sub>B</sub>のみONの場合は“2”が表示され、それぞれ両方ともONの場合には“3”が表示されます。

INP命令はI/Oの入力命令で、入力したデータは、ここでは変数INに入るために、プログラムによる演算や比較処理ができます。

次に出力部のランプをON/OFFする場合には、OUT命令を使用します。

```
OUT &HDO, &H1
```

をキーボードから入力するとランプは点灯します。消灯する場合は、

```
OUT &HDO, 0
```

を入力します。以上の入出力命令を使用した簡単な制御プログラムの例を、リスト1-1に示します。

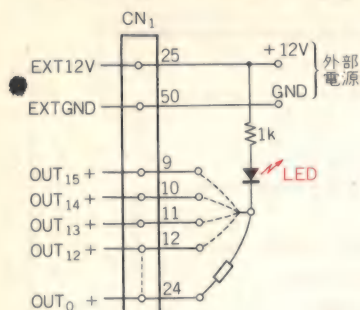
プログラムを実行すると、画面上にDATA=2 5と表示され、入力SW<sub>A,B</sub>のON/OFFに応じたデータが表示されます。DATA=1の場合には、ランプは数秒間入力に変化するまで点灯し、DATA=2の場合は数秒間点滅し、DATA=3が入力されるとランプは消灯しプログラムは終了します。

## 1-3 入出力ポートのチェック

入力ポートのチェックは、入力を開放にし、次のプログラムを実行します。



〈図1-4〉 出力ポートのチェック回路



**A=INP(&HDO):PRINT A**

Aのデータ255, すなわち16進でFFが表示され、入力をすべて“ON”にした場合は0が表示されます。以上の値が表示されれば入力ポートは正常です。

出力ポートのチェックは、出力インターフェース側を開放にし、ch<sub>0</sub>~ch<sub>15</sub>の出力状態をLS273, ラッチの出力部にテストをあててチェックします。ch<sub>0</sub>~ch<sub>7</sub>に相当するラッチは回路図上2CのLS273になり、テストを電圧レンジにし⊖をGNDに接続し、⊕をラッチの出力側にあて、次のプログラムを実行します。

**OUT &HDO, 0**

この場合、出力のデータはすべて“0”になり、ラッチのすべての出力電圧は約0.6Vになります。次にすべての出力データが“1”になることをチェックします。

**OUT &HDO, &HFF**

を実行すると、ラッチのすべての出力電圧は約3.8Vになります。ch<sub>8</sub>~ch<sub>15</sub>に相当するラッチは回路図上2Dになり、次のプログラムにより同様のチェックを行います。

**OUT &HD2, 0**

**OUT &HD2, &HFF**

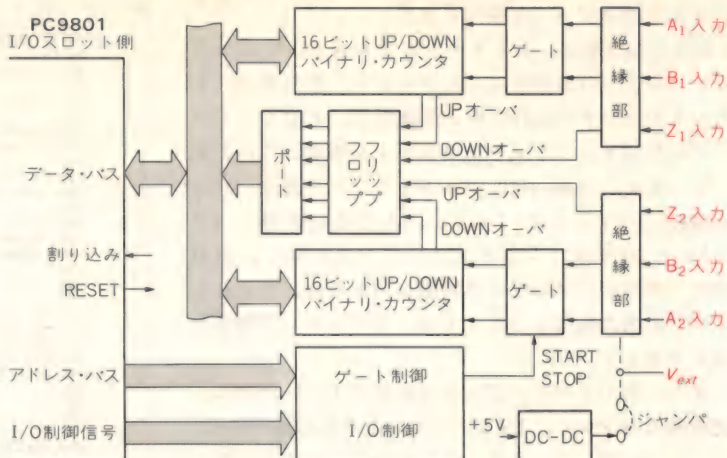
2Cのラッチ出力と同様の電圧が確認できれば、ラッチの出力部まではch<sub>0</sub>~ch<sub>15</sub>すべて正常に動作していることになります。

最終チェックは、CN<sub>1</sub>の25ピンと50ピンに外部電源より12Vを供給し、図1-4のような測定回路によりチェックを行います。この回路は、出力がONのビットだけ点灯することで確認できます。

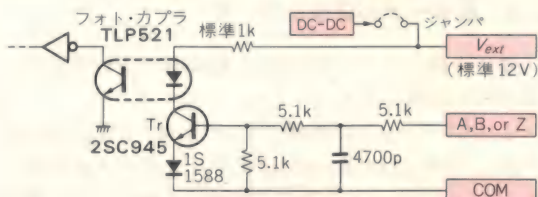
## ②アップ/ダウン・カウンタ・ボードの製作

パソコンで位置、回転数、流量などを測定するときには、カウンタを使うことが多いようです。ここでは

〈図2-1〉 カウンタ・ボードのブロック図



〈図2-3〉 入力の電圧レベルに自由度をもたせる回路構成としての入力回路(絶縁部)の詳細



〈表2-1〉 入力信号モード設定

ジャンパ	設定	入力モード	カウンタ
JP1~1	UP側	単相入力	チャンネル 1
	CW側	2相入力	
JP1~2	DN側	単相入力	
	CCW側	2相入力	
JP2~1	UP側	単相入力	チャンネル 2
	CW側	2相入力	
JP2~2	DN側	単相入力	
	CCW側	2相入力	

FA, LAに適した汎用の2チャンネル16ビット・アップ/ダウン・カウンタ・ボードを紹介します。

### 2-1 本ボードの仕様と構成

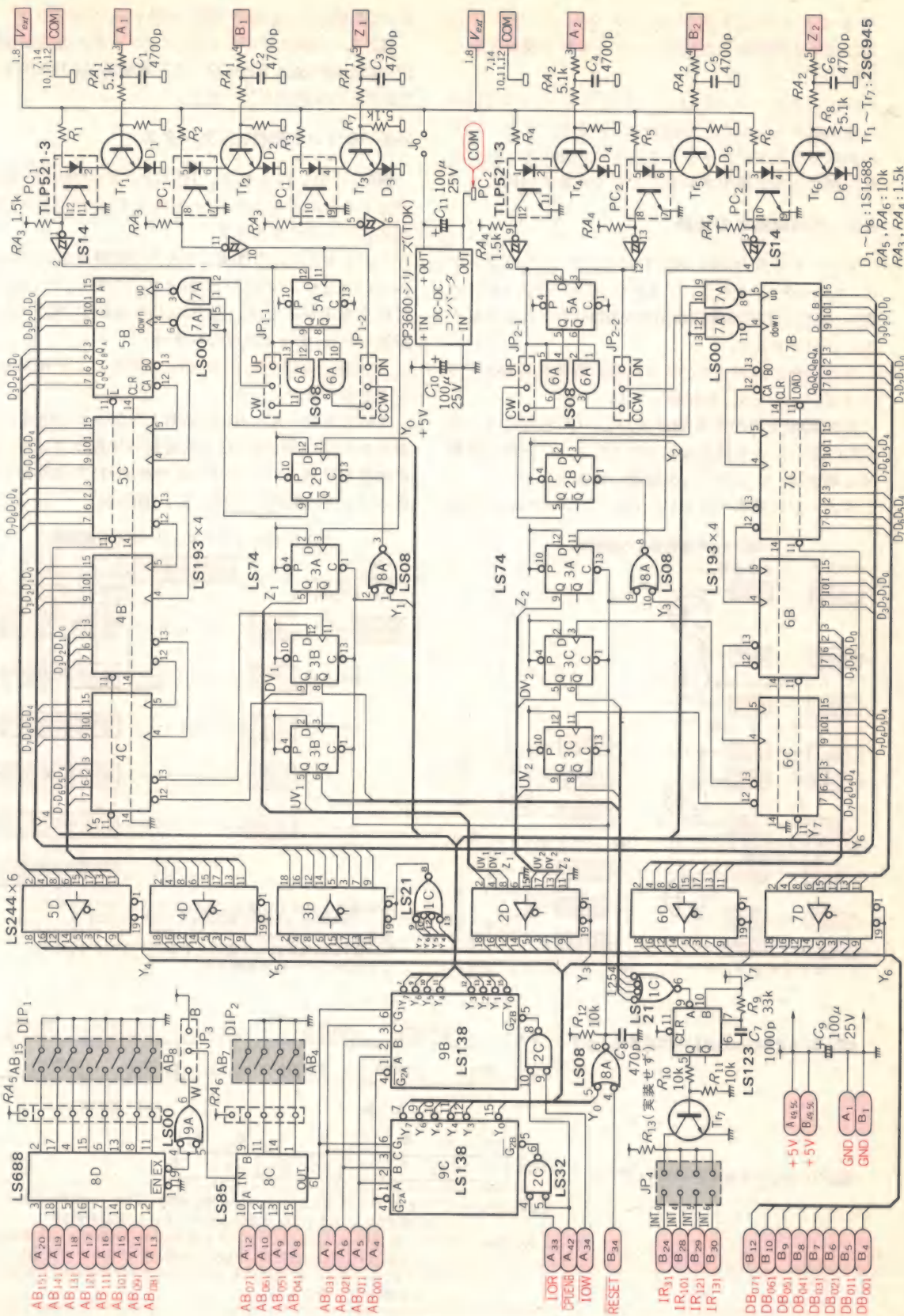
図2-1に本機のブロック図を、図2-2に全回路図を示します。

入力部は図2-3に示すようにフォト・カプラによる絶縁(耐圧AC500V)となっており、基板上に絶縁型のDC-DCコンバータを搭載して入力側電源と絶縁できるようになっています。

また、外来ノイズを除くためのローパス・フィルタも入っており、フォト・カプラの応答速度と相まって約10kHzまでカウントすることができます。

入力信号の形式は、通常の単相信号と、インクリメンタル型のロータリ・エンコーダに使用される2相信号をジャンパで切り替えられます(表2-1)。計数部は

〈図2-2〉カウンタ・ボードの回路





プリセットもできるアップ/ダウン・カウンタ74HC193を4個連結した16ビット・バイナリ構造となっています。

ステータス・フラグは、キャリ、ボロー、およびZ相入力を各チャンネルごとに用意しています。また、いずれかのカウンタがキャリまたはボローを出力したときに、割り込みを発生させることもできます(JP<sub>4</sub>)。

## 2-2 外部機器との接続

本ボードの入力回路(図2-3)には、オープン・コレクタ、エミッタ・フォロワ、TTLレベル(トータム・ボール型)、接点などの外部機器出力を図2-4のように直結することができます。

外部機器と本ボードの入力回路の電源を共通とするときは、 $V_{ext}$ と $V_{CC}$ を接続します。

外部電源を使用する場合は、 $V_{ext}$ -COM間に5〜24Vを印加し、また基板上にDC-DCコンバータを搭載する場合は、ジャンパJ<sub>0</sub>を接続します。

本ボードに搭載できるDC-DCコンバータは、TDK

製のCP3601/3604/3607(電圧が異なる)などです。

図2-5に一般的なロータリ・エンコーダ、小野測器製SP402Z, RP432Z, RP862Z, 立石電機製E6D-CWZ1Eなどとの接続例を示します。

## 2-3 カウント動作, ステータス

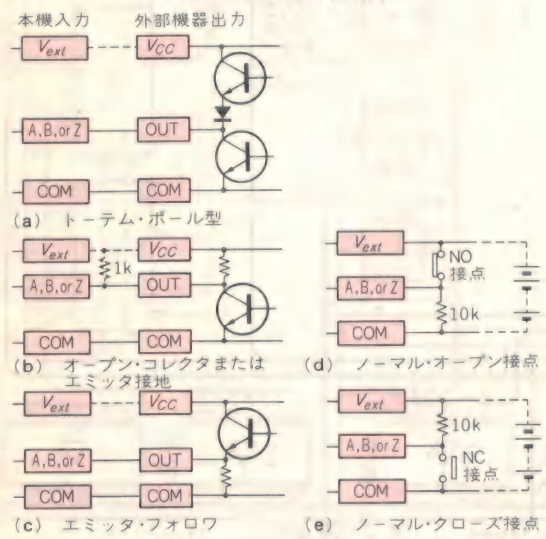
表2-1に示したとおり、JP<sub>1</sub>(JP<sub>2</sub>)をUP側, DN側に設定すると、チャンネル1(チャンネル2)のカウンタが単相入力モードになります。

このときA入力で加算、B入力で減算カウントが、各入力の立ち下がりで行われます。ただし、一方の入力が立ち上がる時、他方の入力は必ず“L”(入力開放も含む)でなければなりません。

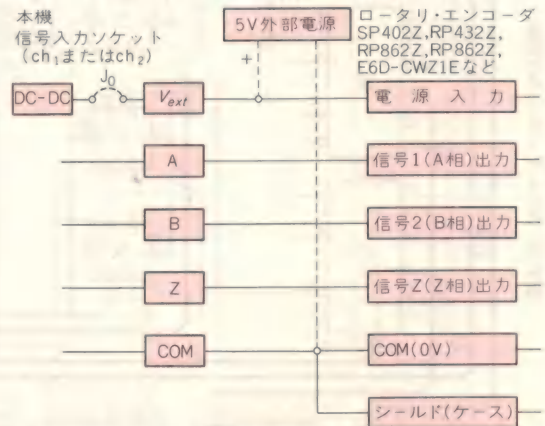
JP<sub>1</sub>(JP<sub>2</sub>)をCW側, CCW側に設定すると2相入力モードとなります。

このときは、A入力信号がB入力信号から90度だけ遅れたらCW(時計回り)方向回転=加算カウント、逆に90度だけ進んだらCCW(反時計回り)方向回転=減算カウントの動作が行われます(図2-6)。

〈図2-4〉 外部機器との接続例

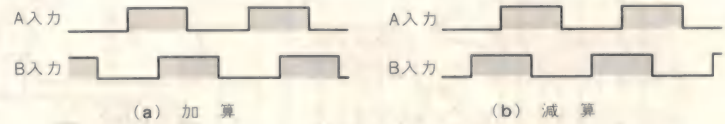


〈図2-5〉 ロータリ・エンコーダとの接続例



外部電源を使用する場合は基板上のジャンパJ<sub>0</sub>を切り、 $V_{ext}$ -COM間に+5V(容量200mA以上)を印加する。基板上に+5V 250mA出力のDC-DCコンバータ(CP3601 TDK製)を追加実装して電源とすることもできる。このときはジャンパJ<sub>0</sub>を接続する

〈図2-6〉 AとBの入力波形の加減算のようす



〈図2-7〉 カウンタのステータス・データ

	B <sub>7</sub>	B <sub>6</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>
MSB	UV <sub>1</sub>	DV <sub>1</sub>	Z <sub>1</sub>	0	UV <sub>2</sub>	DV <sub>2</sub>	Z <sub>2</sub>	0

UV<sub>1</sub>(UV<sub>2</sub>): “1”のとき、チャンネル1(2)のカウンタがCARRY(UPオーバー信号)を出したことを示す(キャリはカウンタ値65,535から0になるとき発生)  
 DV<sub>1</sub>(DV<sub>2</sub>): “1”のとき、チャンネル1(2)のカウンタがBORROW(DOWNオーバー信号)を出したことを示す(ボローはカウンタ値0から65,535になるとき発生)  
 Z<sub>1</sub>(Z<sub>2</sub>): “1”のとき、チャンネル1(2)のZ信号入力が入力有効となった(立ち上がりエッジを検出)ことを示す

なお、加算カウントはA入力の立ち下がりで、減算カウントはB入力の立ち下がりで行われます。ステータスは図2-7に示す構造の1バイト・データで、ひとたびセットされたフラグは、後述のリセット操作を行わない限りもとにもどりません。

## 2-4 CPUとのインターフェース

本ボードの制御およびデータの読み書きは、すべてI/O命令により行います。I/Oアドレスは、上位8ビットをディップ・スイッチDIP<sub>1</sub>で、中位4ビットをDIP<sub>2</sub>で割り付け、下位4ビットは固定となっています。

なお、JP<sub>3</sub>はPC9801Fまでの旧機種をBASICのみで使用するとき、I/Oアドレスが下位8ビットしか有効でないため、B側に設定して上位8ビットを無視するためのジャンプです。

しかもI/Oアドレスは空きが少なく、中位4ビットは通常Hex“D”しか使用できません。

PC9801M以降の機種では、BASICでも上位8ビットをHex“00”～“7F”まで使用できますから、JP<sub>3</sub>をW側に設定し、Hex4桁のI/Oアドレスを使用するとよいでしょう。

なお、機械語プログラム中では、新旧に関係なくHex4桁のI/Oアドレスを使用することができます。

## 2-5 操作方法、プログラム例

本ボードの具体的操作は、表2-2に示したI/Oポー

トを読み書きすることです。

例えば、リセット操作をBASICにより記述すると、

```
C = INP(&HDO)
```

となります。このとき両カウンタのゲートは閉じられ、ステータス・フラグがすべてクリアされます。

ただし、カウンタはクリアされません。なお、変数Cには何の意味もなく、I/O制御信号によってリセット動作が行われます。同様にチャンネル1のカウンタ・ゲートを開くには、

```
OUT &HDO, m
```

とします。mは0～255の任意の数で、やはり意味がありません。

カウンタのクリアは、ゼロを書き込むことにより行います。すなわち、

```
OUT &HD8, 0
```

でカウンタの下位8ビットに、また、

```
OUT &HDA, 0
```

で上位8ビットにそれぞれゼロを書き込むことができます。

カウント値は、

```
DH = INP(&HDA)
```

で上位8ビット、

```
DL = INP(&HD8)
```

で下位8ビットをそれぞれ変数DH、DLに読み込むことができます。

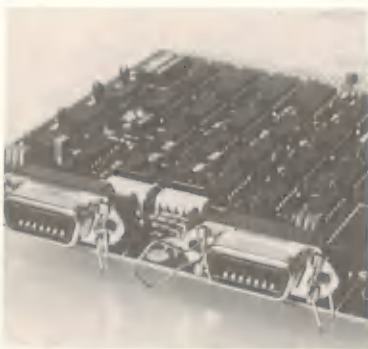
メイン・フローをBASICで、I/O制御ルーチンを機

〈表2-2〉I/Oポートの機能とアドレス割り付け(HEX表示)

出力ポート機能	I/Oアドレス	入力ポート機能
チャンネル1 GATE ON (START)	××D0	フラグ・クリア、制御回路リセット
チャンネル1 GATE OFF (STOP)	××D2	なし
チャンネル2 GATE ON (START)	××D4	なし
チャンネル2 GATE OFF (STOP)	××D6	ステータス・フラグ読み込み
チャンネル1 下位8ビット書き込み	××D8	チャンネル1 下位8ビット・データ読み込み
チャンネル1 上位8ビット書き込み	××DA	チャンネル1 上位8ビット・データ読み込み
チャンネル2 下位8ビット書き込み	××DC	チャンネル2 下位8ビット・データ読み込み
チャンネル2 上位8ビット書き込み	××DE	チャンネル2 上位8ビット・データ読み込み

\* “××”はDIP<sub>1</sub>による上位8ビット・アドレス(AB<sub>151</sub>-081)設定値。Hex 00～7F

\* DIPスイッチの各ビットはONが“0”、OFFが“1”に対応する



〈写真2-1〉カウンタ入力コネクタ付近

# 最新 マイコン周辺LSI規格表

B5判 224頁 定価1200円

CQ出版社

本書は、8ビット/16ビットCPUの周辺に使われるLSIを、4～8頁単位でわかりやすくまとめたものです。

掲載LSI

割り込みコントローラ (MC6828, 8259A) DMAコントローラ (MC6844, Z8410/DMA) タイマ/カウンタ (MC6840, Z8036/CIO, 8253, Z8430/CTC,  $\mu$ PD71011C) リアルタイム・クロック (MC146818, MSM6242RS, TC8250P) CRTコントローラ ( $\mu$ PD3301, HD63484, MC6845/HD46505,  $\mu$ PD7220A, 8275) フロッピー・ディスク・コントローラ/データ・セパレータ (FD179X-02, MC6843,  $\mu$ PD765AC, FDC9216, SED9420C) 直列インターフェース (IM6402A/3A, MC6850,  $\mu$ PD7201A, Z8030/SCC, INS8250, 8251A, Z8440～8442/SIO, LH8572) 並列インターフェース (MC6821, 8216/8226, 8255A, 8279, Z8420/PIO) GPIBインターフェース (MC68488,  $\mu$ PD7210, 8291A, 8292, TMS9914)



```

40 ' [ DIP2 を "D" , JP3 を W (ワート・モト) にする ]
50 '
1000 CLEAR ,&H1FA0 : IH=&H1F:IL=&HAA0 ' 割り込み処理ルーチンの先頭アドレス
1010 INTA=256:IH+IL ' RAM を増設しているかまたは日本語BASICシステムを使用している
' 場合には "1FA0" を "3FA0" などとする(2箇所)
1020 '
1030 ' ----- ハ*ラマータ の設定 -----
1040 INTRPT=6 ' 割り込み line [INT6] ; INT4,5 ならば INTRPT=4,5 ( JP4 )
1050 UPBYT#=00 " ' I/O アドレス 上位 1A*イト の設定 ( &H00-FF ; DIP1 )
1060 IDT1X=0 ' ch 1 カウンタ の初期値 } 0.5 キー でセットされるカウンタ
1070 IDT2X=0 ' ch 2 カウンタ の初期値 }
1080 '
1090 ' ----- 割り込み処理の初期設定 -----
1100 IF INTRPT=6 THEN I=&H54:MMSK=&H7F:MIRL=&H67:SMSK=&HDF:SIRL=&H65:GOTO 1110
ELSE IF INTRPT=5 THEN I=&H50:MMSK=&H7F:MIRL=&H67:SMSK=&HDF:SIRL=&H64:GOTO 1110
ELSE IF INTRPT=4 THEN I=&H48:MMSK=&H7F:MIRL=&H67:SMSK=&HDF:SIRL=&H62 ELSE END
1110 DEF SEG=0 ' 割り込みサービス・ルーチンの先頭アドレス ( low,high )
1120 POKE I,&H0 :POKE I+1,&H0 ' オフセット(&h) = 00 , 00
1130 POKE I+2,IL :POKE I+3,IH ' セタメント(&h) = 00 , 1E
1140 MPIC=INP(2) :MSET=(MPIC AND MMSK):OUT &H2,MSET:OUT 0,MIRL ' 割り込みコントローラ
1150 SPIC=INP(&HA):SSET=(SPIC AND SMSK):OUT &HA,SSET:OUT 8,SIRL ' の前処理
1160 '
1170 DEF SEG=INTA ' RAMエリア、機械語の先頭の宮、
1180 DIM DT1X,DT2X,FLGX,UV1X,DV1X,Z1X,OV1X,UV2X,Z2X,OV2X ' 機械語サブルーチンとリンクするデータ、フラグ
1190 RESTORE 2060:FOR I=0 TO &H16F:READ X$:POKE I,VAL("&h"+X$):NEXT I
1200 X=VARPTR(DT1X,1):I=INT(X/256):POKE &H9,X-I+256:POKE &HA,I ' データ、フラグの
1210 X=VARPTR(DT1X,1):I=INT(X/256):POKE &HE,X-I+256:POKE &HF,I ' アドレスのセット
1220 RESTORE 2020:READ RD,WR,CLR,START1,STOP1,START2,STOP2 ' 機械語のサブルーチン
1230 UB=VAL("&h"+UPBYT$) ' I/O アドレス 上位バイトのセット
1240 RESTORE 2030:FOR I=1 TO 8:READ X$:POKE VAL("&h"+X$),UB:NEXT I
1250 DIM X$(5) : RESTORE 2040:FOR I=1 TO 5:READ X$(I):NEXT I
1260 '
1270 ON KEY GOSUB *START1,*START2,*STOP1,*STOP2,*CNTCLR
1280 FOR I=1 TO 5 : KEY(I) ON : NEXT I
1290 ON HELP GOSUB *ENDPROC : HELP ON
1300 WIDTH 80,25 : CONSOLE 0,25,0,0 : LOCATE 60,24 : PRINT "HELP:キ- ー end";
1310 LOCATE 27,3 : PRINT "[ channel 1 ] [ channel 2 ]"
1320 LOCATE 12,6 : PRINT "counter" : LOCATE 12,9 : PRINT "carry (UV)"
1330 LOCATE 12,11 : PRINT "borrow(DV)" : LOCATE 12,13 : PRINT "signal (Z)"
1340 LOCATE 12,16 : PRINT "over speed" : LOCATE 3,21
1350 COLOR 4:FOR I=1 TO 5:LOCATE 10+I-5,24:PRINT X$(I):NEXT I :COLOR 0
1360 GOTO *JOB
1370 '
1380 ' ----- basic language subroutine -----
1390 *DISP
1400 LOCATE 31,6:IF DT1X=0 THEN PRINT USING"#####":DT1X
ELSE PRINT USING"#####":65535!+DT1X
1410 LOCATE 53,6:IF DT2X=0 THEN PRINT USING"#####":DT2X
ELSE PRINT USING"#####":65535!+DT2X
1420 LOCATE 32,9 : PRINT UV1X : SPACE$(19) : UV2X
1430 LOCATE 32,11 : PRINT DV1X : SPACE$(19) : DV2X
1440 LOCATE 32,13 : PRINT Z1X : SPACE$(19) : Z2X
1450 LOCATE 32,16 : PRINT OV1X : SPACE$(19) : OV2X
1460 RETURN
1470 *START1 : CALL START1 : RETURN
1480 *START2 : CALL START2 : RETURN
1490 *STOP1 : CALL STOP1 : RETURN
1500 *STOP2 : CALL STOP2 : RETURN
1510 *CNTCLR : CALL CLR : CALL WR(IDT1X,IDT2X) : RETURN
1520 *ENDPROC
1530 CALL STOP1 : CALL STOP2
1540 OUT &H2,MPIC : OUT &HA,SPIC ' 割り込みコントローラの後処理
1550 CONSOLE 0,25,1,0 : KEY OFF : HELP OFF : LOCATE 0,22 : END
1560 '
2000 ' .... machine code subroutine & tuning data ....
2010 ' < ドライバ・ファルテン tuning data > -----
2020 DATA 0,&HE0,&H120,&H130,&H140,&H150,&H160
2030 DATA 7, 59, E7, 123, 133, 143, 153, 163
2040 DATA "ch1 start", "ch2 start", "ch1 stop", "ch2 stop", "clr & set"
2050 ' < 機械語の サブ ルーチン > -----
2060 DATA 06,56,50,51,52,53,B6,00,B8,00,1E,8E,C0,B8,20,10 ' < RD >
2070 DATA 89,C6,B4,02,B2,DA,EC,88,C1,B2,DE,EC,88,C5,B2,D8
2080 DATA EC,88,C3,B2,DC,EC,88,C7,B2,DA,EC,38,C8,74,08,B1
2090 DATA 01,FE,CC,74,2C,EB,DD,B2,DE,EC,38,E8,74,08,B1
2100 DATA FE,CC,74,1D,EB,CE,87,DA,88,D0,88,CC,26,89,04,88
2110 DATA F0,88,EC,26,89,44,08,BA,D6,00,EC,26,88,44,10,B1
2120 DATA 00,B4,01,B5,00,80,E9,00,75,4A,26,88,6C,18,A8,80
2130 DATA 74,04,26,88,64,18,26,88,6C,20,A8,40,74,04,26,88
2140 DATA 64,20,26,88,6C,28,A8,20,74,04,26,88,64,28,26,88
2150 DATA 6C,38,A8,08,74,04,26,88,64,38,26,88,6C,40,A8,04
2160 DATA 74,04,26,88,64,40,26,88,6C,48,A8,02,74,04,26,88
2170 DATA 64,48,EB,1A,26,88,6C,30,F6,C1,02,74,04,26,88,64,50,5B,5A
2180 DATA 30,26,88,6C,50,F6,C1,02,74,04,26,88,64,50,5B,5A
2190 DATA 59,58,5E,07,CF,90,90,90,90,90,90,90,90,90,90,90
2200 DATA 06,56,50,53,51,52,B6,00,B9,03,00,B2,D0,EC,8B,47 ' < WR >
2210 DATA 06,8E,C0,8B,77,04,26,8B,04,B2,D8,EE,88,E0,B2,DA
2220 DATA EE,8B,47,02,8E,C0,8B,37,26,8B,04,B2,DC,EE,88,E0
2230 DATA B2,DE,EE,EC,D6,5A,59,5B,58,5E,07,CF,90,90,90,90
2240 DATA 50,52,B6,00,B2,D0,EC,5A,58,CF,90,90,90,90,90,90 ' < CLR >
2250 DATA 50,52,B6,00,B2,D0,EE,5A,58,CF,90,90,90,90,90,90 ' < START1 >
2260 DATA 50,52,B6,00,B2,D2,EE,5A,58,CF,90,90,90,90,90,90 ' < STOP1 >
2270 DATA 50,52,B6,00,B2,D4,EE,5A,58,CF,90,90,90,90,90,90 ' < START2 >
2280 DATA 50,52,B6,00,B2,D6,EE,5A,58,CF,90,90,90,90,90,90 ' < STOP1 >
2290 '
2300 '
2310 '
2320 '
2330 '
2340 ' *****
2350 ' ..... シ ョ フ * ルーチン .....
2360 ' *****
2370 '
2380 *JOB
2390 CALL CLR
2400 DT1X=1024 : DT2X=3000 ' カウンタのデータ
2410 CALL WR(DT1X,DT2X) ' カウンタのデータの書き込み
2420 CALL START1 : CALL START2 ' スタート(gate open)
2430 CALL RD ' カウンタ、フラグの読み込み
2440 GOSUB *DISP
2450 GOTO 3090

```

<リスト2-1>

I/O制御部に機械語を用いた  
コントロール・プログラム例

## 〈リスト2-2〉

BASICで記述したコントロール・プログラム例

```

10 '
20 '
30 '
40 ' [ DIP2 を "D" , JP3 を B (ハイト・モート) にする ]
50 '
100 CLS : CONSOLE 3,22,0,0 : PRINT SPACE$(60); "HELP" キーで stop
110 PRINT "channel 1 channel 2 flag": LOCATE 0,3
120 ON HELP GOSUB *MENTHELP : HELP ON
130 '
140 C=INP(&HD0) ' リセット
150 DL1=8:DH1=1 : DL2=2:DH2=1 ' カウンタの初期値のセット
160 GOSUB *WR
170 OUT &HD0,0 : OUT &HD4,0 ' スタート (gate open)
180 '
190 GOSUB *RD
200 GOSUB *DISP
210 GOTO 190
220 '
230 *WR '..... カウンタ・データの書き込み
240 FOR I=1 TO 3
250 C=INP(&HD0)
260 OUT &HD8,DL1 : OUT &HDA,DH1
270 OUT &HDC,DL2 : OUT &HDE,DH2
280 NEXT I
290 RETURN
300 '
310 *RD '..... カウンタ・データの読み込み
320 DL1=INP(&HD8) : DH1=INP(&HDA)
330 DL2=INP(&HDC) : DH2=INP(&HDE)
340 RETURN
350 '
360 *DISP '..... 表示
370 C1=256*DH1+DL1 : C2=256*DH2+DL2
380 FLG$=HEX$(INP(&HD6))
390 PRINT USING "#####&&" ;C1,C2,FLG$
400 RETURN
410 '
420 *ENDING
430 OUT &HD2,0 : OUT &HD6,0 ' ストップ (gate close)
440 LOCATE 0,23 : END
450 '
460 *MENTHELP
470 HELP OFF : CONSOLE 0,25,1,0 : RETURN *ENDING

```

〈表2-3〉 リスト中の主要サブルーチン一覧

機械語サブルーチン名	内 容
CLR	両カウンタのゲートを閉じ、ステータス・フラグをクリアする
START1	チャンネル1のカウンタ・ゲートを開く
START2	チャンネル2のカウンタ・ゲートを開く
STOP1	チャンネル1のカウンタ・ゲートを閉じる
STOP2	チャンネル2のカウンタ・ゲートを閉じる
RD	両カウンタ値、ステータス・フラグを読み込み、指定変数に代入する
WR	指定変数(DT1%, DT2%)の値を両カウンタに書き込む

BASICサブルーチン名	内 容
* DISP	読み込んだカウンタのデータ、ステータス・フラグをCRTに表示する
* CNTCLR	両カウンタのゲートを閉じ、ステータス・フラグおよび両カウンタをクリアする
* ENDPROC	両カウンタのゲートを閉じ、割り込みコントローラの後処理、CRTを初期化する

械語で記述した例をリスト2-1に、またBASICのみで記述したプログラム例をリスト2-2に示します。

いずれも操作内容を行中にコメントで記してありますから詳細は省略します。とくに前者のプログラム例では、操作の種類ごとに表2-3のようなサブルーチンを用意し、メイン・フロー(3000行以下)では、所望のサブルーチンを呼ぶだけで実用的なプログラムができます。

## ③ 4 ch直流SSR出力ボードの製作

最近では、リレーの代わりにSSR(ソリッド・ステート・リレー)が利用され、コンピュータによる電力制御

が行われてきています。

SSRは機械的接点をもたないために、接点のアーカ・ノイズ、接触不良などがなく、信頼性が高いといえます。また、SSRの入出力部は完全に絶縁されているため、外部の周辺ノイズ、誘導ノイズなどの影響に強く、安定に動作します。

駆動は、TTL出力で十分に動作することができるため、取り扱いが簡単で、FA制御用に多く使用されています。

ここで製作するボードは、直流負荷用SSRを4個搭載した、4ch直流SSRインターフェース・ボードで、PC9801シリーズの拡張I/Oスロットに挿入して使用するものです。



3-1 SSR出力ボードの仕様と構成

図3-1に、本ボードのブロック図を、図3-2には全回路を示します。

I/Oアドレス・デコーダは、 $A_1 \sim A_{15}$ の15アドレスを使用し、ボード上のディップ・スイッチにより、先頭アドレスを任意に設定することができます。バスのデータは、 $D_0 \sim D_{15}$ の16ビットですが、内部で使用するデータ信号は、偶数アドレス時は $D_0 \sim D_3$ 、奇数アドレス時は $D_4 \sim D_{11}$ のそれぞれ4ビットを使用します。

4ビットのデータ信号は、LS175のラッチに入力され、出力 $Q/\bar{Q}$ 各4ビットの出力になります。各 $Q/\bar{Q}$ は $J_2 \sim J_5$ を通り、7404をドライバとして、各SSRに入力されます。 $J_2 \sim J_5$ の切り替えによって、ボード電源投入時の各SSRの出力論理を設定することが可能です。

SSRの取り扱う電力は大きいため、デジタル回路にノイズの影響がないように、SSR部は信号パターンの強化、配置などが考慮されています。外部との接続には、大電力に耐える圧着端子型の端子台を使用し、信頼性を上げており、外部との接続が簡単に行えるようになっています。

SSRの出力部は、四つの出力チャンネルを独立して制御することができます。被制御直流電圧は200V(最大)、被制御直流電流は1A(定格)、1秒サージ電流は2A(最大)、出力応答時間は2.5ms(最大)であり、最大200VAの電力を制御することができます。表3-1にSSRの特性を示します。

3-2 本ボードの操作

操作の例としては、PC9801のBASICのI/O命令を用いて行います。このボードのI/O先頭アドレスは、ディップ・スイッチSW<sub>1</sub>下位( $A_1 \sim A_7$ )、ディップ・スイッチSW<sub>2</sub>上位( $A_8 \sim A_{15}$ )によりアドレスを設定します。しかし、上位8ビットは機械語を使用した場合以外は、16進数で“00”を設定し、下位7ビットでデコードします。

ここではD0h番地を先頭アドレスとしたときの例として表3-2にアドレスと機能を示します。次に、本ボードとN<sub>88</sub>BASICで記述した例により、図3-3の参考回路例で説明します。接続にあたっては、回路の中で高い電圧を使用するので、感電、漏電、ショートなどに十分に注意し、電源の極性を正しく合わせ配線することが必要です。

今、チャンネル1にDCモータが接続されています。モータをON/OFFする場合、次のようにキーボードから入力します。

OUT & HDO, 1.....(1)

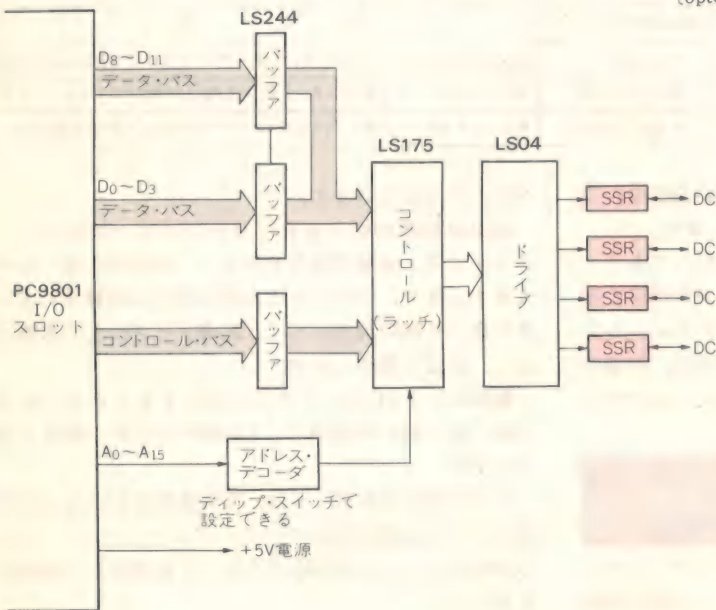
(1)を実行するとモータはONになり回転します。

OUT & HDO, 0.....(2)

(2)を入力するとモータは停止します。

四つのチャンネルすべてを、画面上で出力の状態を見ながら操作することができるプログラム例をリスト3-1に示します。このプログラムを実行すると、画面上に $ch_1 \sim ch_4$ に対応する白色の“○”印が4個表示されます。キーボードから“1”を入力すると、 $ch_1$ の

＜図3-1＞ SSRボードのブロック図

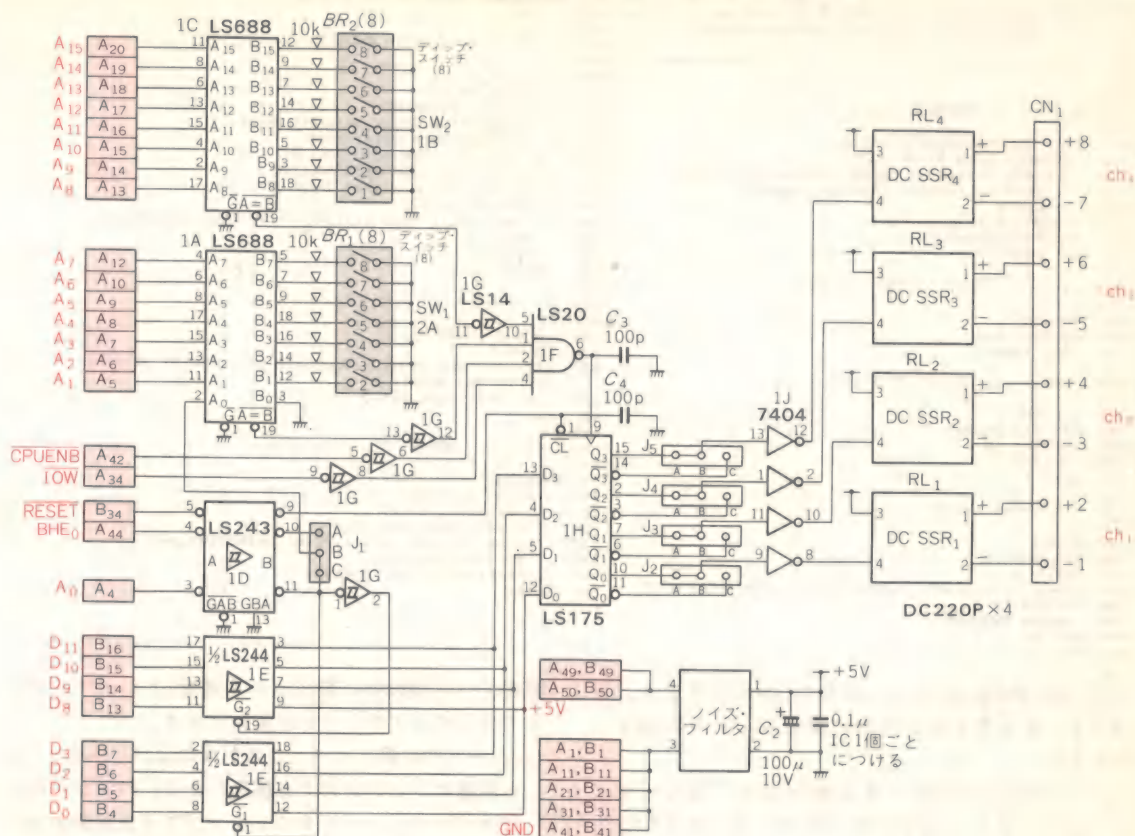


＜表3-1＞ SSRの特性表(DC220P)

(opto22製、山田工業㈱電子部扱い ☎03(778)2811)

特性項目	値
定格主回路電圧 (max)	200 V <sub>DC</sub>
定格主回路動作電圧	4~200 V <sub>DC</sub>
定格主回路電流 (typ)	1 A
1秒サージ電流	2 A
開路時ブロック電圧	200 V <sub>DC</sub>
閉路時電圧降下	1.5 V
開路時もれ電流	1 mA
入力電圧	3~32 V <sub>DC</sub>
入力ピックアップ電圧	3 V <sub>DC</sub>
入力ドロップアウト電圧	1 V <sub>DC</sub>
入力抵抗	1 kΩ
ターンオン時間	0.1 ms
ターンオフ時間	0.75 ms
絶縁耐圧	4 kV(RMS)
使用可能動作温度範囲	-40~+100℃

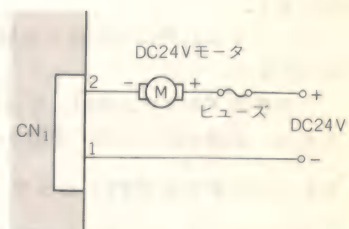
〈図3-2〉 4チャンネルの直流SSRをコントロールするボードの回路



〈表3-2〉  
I/Oアドレスの設定

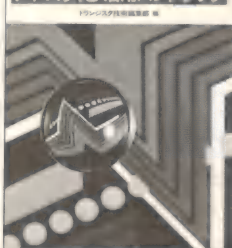
	データ (下位 4 ビット)			
先頭アドレス	0	1	2	3
DOH	ch1	ch2	ch3	ch4

〈図3-3〉  
モータを接続した例



絶賛発売中!

トランジスタ技術 増刊  
ハードウェア・デザイン・シリーズ  
アナログIC活用ハンドブック



CQ出版社

## ハードウェア・デザイン・シリーズ

★電子回路部品活用ハンドブックにつづく第2弾!

# アナログIC

トランジスタ技術 増刊

2色刷

B5判 320頁  
定価 1,800円  
送料 300円

# 活用ハンドブック

内容

- (1) プロローグ
- (2) OPアンプの基本機能
- (3) OPアンプと線形回路
- (4) 非線形の演算回路
- (5) フィルタ回路
- (6) 発振回路およびV-Fコンバータ
- (7) D-Aコンバータ
- (8) A-Dコンバータ
- (9) 電源用IC
- (10) スイッチング・レギュレータ



```

10 *****
20 * AB98-13A 4ch SSR BOARD SAMPLE PROGRAM *
30 *
40 *****
50 ?
60 ?
70 ? ----- 初期設定 -----
80 ?
90 WIDTH 80,25:SCREEN 0,0:CLS 3
100  *ALLOFF
110  LOCATE 4,1:PRINT "AB98-13A 4CH SSR BOARD"
120  LOCATE 40,1:PRINT "SAMPLE PROGRAM"
130  COLOR 5
140  LOCATE 5,6:PRINT "CH"
150  D=0:P=0:ADDR=&HDO
160  DIM N(25)
170  *
180  *----- 出力すべてOFF -----
190  *
200  *ALLOFF
210  A=0:I1=1:P=7
220  GOTO *START
230  *
240  *----- 出力すべてON -----
250  *
260  *ALLON
270  A=15:I1=25:P=2
280  FOR I=1 TO 4
290    N(I)=1
300  NEXT I
310  *
320  *-----スタート-----
330  *
340  *START
350  OUT ADDR,A
360  COLOR 3
370  LOCATE 0,23:PRINT "output-DATA...&H";HEX$(A);
380  FOR M=0 TO 3
390    GOSUB *NUMBER
400  NEXT M
410  D=1
420  *
430  *-----キー入力データの比較-----
440  *
450  *INDATA
460  FOR I=1 TO 25
470    LOCATE 2,21:PRINT " "
480    COLOR 7:LOCATE 2,21:INPUT N(I)
490  IF N(I)=100 THEN PRINT :END
500  IF N(I)=99 THEN GOTO *ALLOFF
510  IF N(I)=88 THEN GOTO *ALLOFF
520  IF N(I)<1 OR N(I)>4 THEN GOTO 480
530  FOR J=1 TO I-1
540    IF N(I)=N(J) THEN N(J)=N(I-1):P=7:U=-1:GOTO 570
550  NEXT J
560  P=2:U=1
570  M=N(I)-1
580  *
590  *----- ビット出力 -----
600  *
610  GOSUB *NUMBER
620  COLOR 3
630  IF Y=0 THEN A=A*2^X*U:LOCATE 17,23:PRINT HEX$(A)
640  IF U=-1 THEN I=I-2
650  NEXT I
660  *
670  *----- 表示ルーチン -----
680  *
690  *NUMBER
700  X=M MOD 8:Y=M/8
710  X1=8+3*X:Y1=6+5*Y
720  IF D=1 THEN GOTO 740
730  COLOR 5:LOCATE X1,Y1:PRINT USING"##";M+1
740  X2=X1+1:Y2=Y1+2
750  COLOR P:LOCATE X2,Y2:PRINT " "
760  RETURN

```

“0”印が赤色に変化し、出力がONになります。再び“1”を入力すると白色に変化し、出力はOFFになります。

すべての出力をONにする場合には、“99”を入力します。また、OFFにする場合には“88”を入力します。

プログラムの実行を中止する場合は、“100”を入力します。

この参考プログラム例は、もっとも基本的な使用方法であり、制御分野では広く使用されています。

### 3-4 ハードウェアのチェック

プログラムにより、目的のデータがSSRに出力されているかどうかをチェックします。IC 1Jの7404の出力論理をテストなど使用して調べてください。まず、次のプログラムを実行します。

**OUT &HDO, 0**

7404の出力はすべて“1”になり、電圧は約3.8Vになります。次に、

**OUT &HDO, &HOF**

を実行し、4ビットともに“0”，電圧が約0.6Vとなれば正常で、SSRユニットが動作します。IC 1Jの論理が指定データと異なる原因として、**I/Oアドレスの設定が誤っている**ことが考えられます。

## ④ステッピング・モータ制御ボードの製作

パソコンでステッピング・モータを駆動するには、

駆動すべき機械系に要求される速度、トルクなどによって少しずつアプローチが変わります。

① ハード的に最もロー・コストなのは、I/Oポートから直接にトランジスタを駆動したり、パラレル出力付きのシフトレジスタでトランジスタを駆動する方法です。この方法は**ステッピング・モータの動きを直接CPUで制御することになるのでソフトの負担が重くなりますが、利点はロー・コストですから大量生産向きといえるでしょう。**

② 最近はステッピング・モータの直接制御を専用LSIに任せ、**パソコンからは専用コマンドを与えるだけですむ方法が一般的で、この形のコントローラ・ボードが数社から発売されています。**

③ 上記のいずれの場合も、ステッピング・モータの制御に**高速・高トルクが要求されるときは定電流ドライバ回路が必要となるので、市販の専用ドライバを併用する方法が一般的です。**

### 4-1 本ボードの仕様と構成

ここでは専用LSI(アンペール製PPMC101C)を使用した回路を2チャンネル搭載し、I/Oポートを介して専用コマンドやステータス・データを授受する形としました(図4-1)。図4-2に本ボードの全回路図を示します。

ステッピング・モータの駆動には、フォト・カプラ付きのダーリントン・トランジスタTLP573(東芝)を使用して、パソコン側から絶縁しながら、最大で1Aの駆動能力を得ています。この他に、市販のステッピング・モータ・ドライバを使用するときに必要なパル

ス出力、回転方向信号、また、リミットおよび基準点スイッチ入力もフォト・カプラ絶縁としました。

## 4-2 ステッピング・モータ駆動回路

本ボードでは3～5相モータを制御することができます。図4-3にその接続例および計算例を示します。

ステッピング・モータの駆動回路はLR直列回路ですから、その応答は時定数 $L/R$ に依存します。図4-3で $L$ 、 $R_{pm}$ はステッピング・モータに固有の定数で、外付けの直列抵抗 $R_{ex}$ とあわせて時定数が決まりますから、 $R_{ex}$ を大きくするほど応答が速くなります。

その一方、 $R_{ex}$ を大きくするほど規格の電流を流すために必要な電源電圧が高くなり、電力効率が低下しますが、回路が簡単でコストが低いというメリットが

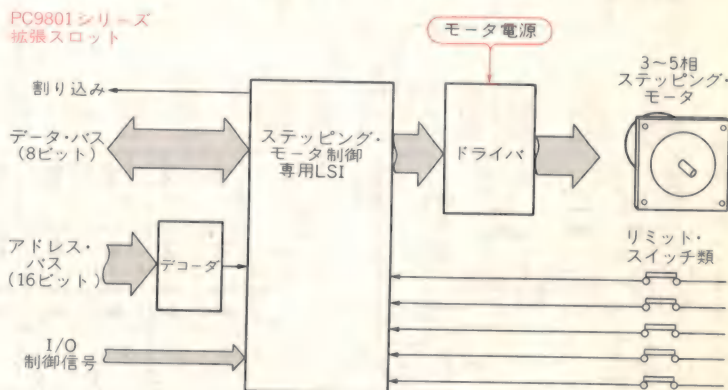
ら小型モータの駆動に適しています。

この回路を使用する場合、ステッピング・モータの種類、負荷状態にもよりますが、約1000pps(ステップ/秒)程度まで回すことができます。より高速運転を行いたいときは、市販の定電流駆動ドライバに本ボードからパルスを送る方法があります(CW, CCW)。この場合、PPMC101Cは5kppsまで制御することができます。

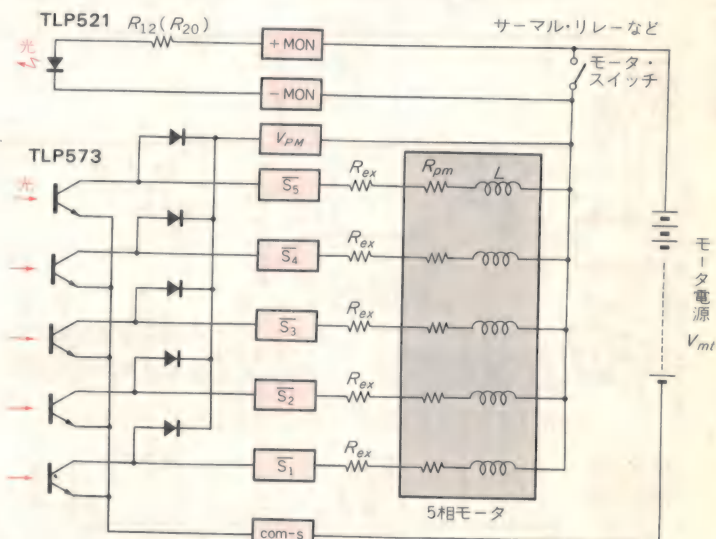
## 4-3 制御系モデル

ステッピング・モータによる代表的な制御系モデルを図4-4に示します。ここでキャリアはステッピング・モータの回転により直線上を移動するものとし、その移動範囲内に制御指標としてのリミット点 $L_1$

〈図4-1〉  
専用LSIによるステッピング・モータ制御



〈図4-3〉  
ステッピング・モータの駆動用電源の接続例

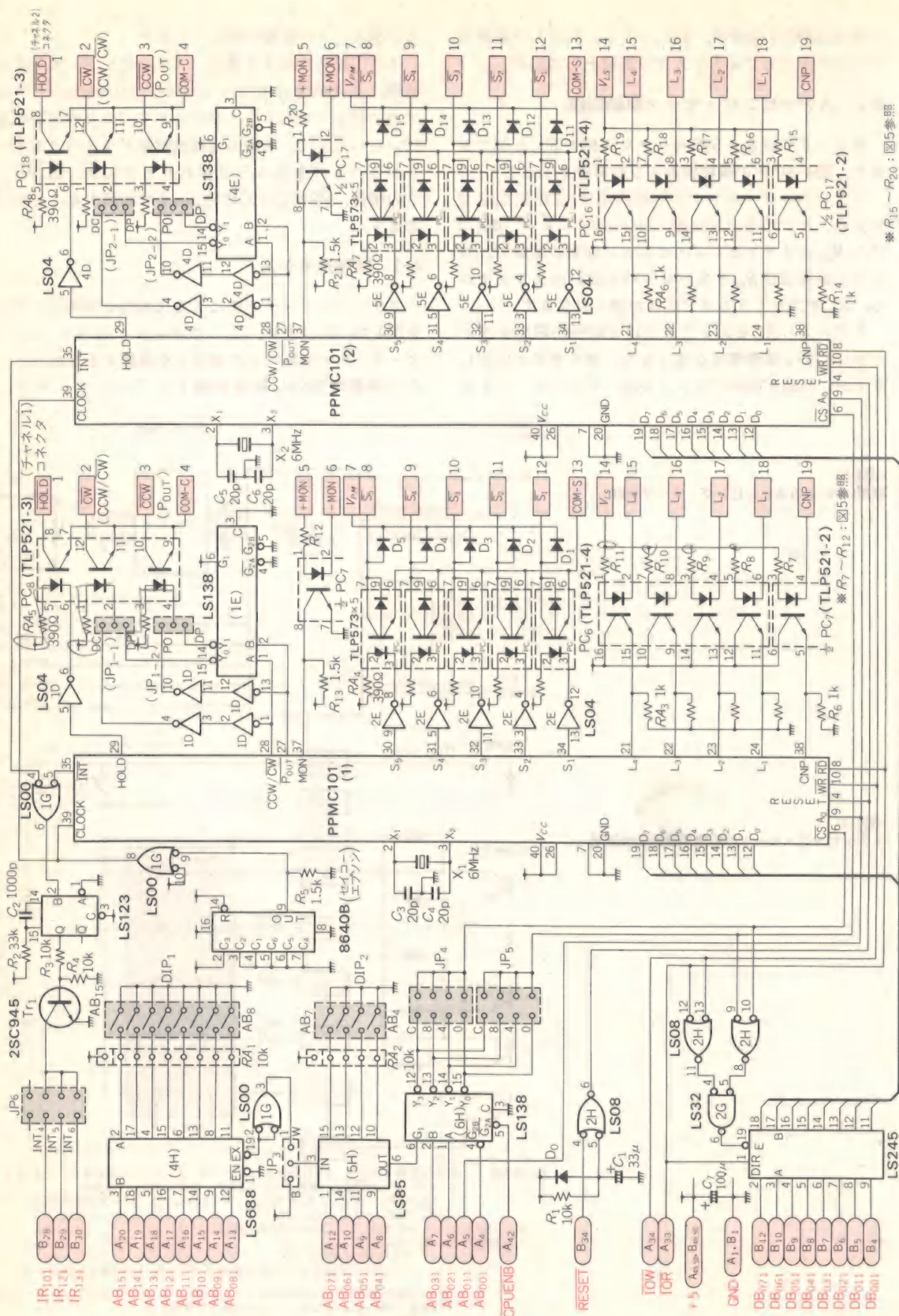


【計算例】 上図の回路で5相モータPH564(オリエンタル・モータ製)を駆動することを考える。  
(このモータの規格は、相電流 $I_{pm}=0.75A$ 、巻線抵抗 $R_{pm}=2.5\Omega$ )  
 $I_{pm}=(V_{mt}-V_{on})/(R_{ex}+R_{pm})$   $V_{on}=1V$ (トランジスタの飽和電圧)  
 $\therefore R_{ex}=(V_{mt}-V_{on})/I_{pm}-R_{pm}$   
 $V_{mt}=5V$ なら、  
 $= (5-1)/0.75-2.5=2.8\Omega$   
消費電力は、 $P_{ex}=I_{pm} \times I_{pm} \times R_{ex}$   
 $=0.75 \times 0.75 \times 2.8$   
 $=1.575W$ (実際は2.5～3倍の余裕をみて、4～5Wの素子を使用する)

モータ電源ON/OFF検出回路のフォト・カプラには、OFFのとき10mA(8～12mA)の電流が流れるように $R_{12}(R_{20})$ を基板上に実装する。  
 $R_{12}(R_{20})=(V_{mt}-2)/(0.01-(R_{pm}+R_{ex})) \approx 300\Omega$



〈図4-2〉 ステッピング・モータ・ドライブ回路



\*  $R_{15} \sim R_{20}$ : ☒ 5参照

ILP521-Z) (7-179471)

～ $L_4$ 、および基準点CNPをマイクロスイッチ、フォト・インタラプタなどを使用して設定します(図4-5)。

#### ▶絶対リミット点 $L_1$ , $L_2$

$L_1$ ,  $L_2$ はCW, CCW各方向のそれ以上動かすことのない位置に設定します。キャリアは $L_1$ ,  $L_2$ まで達すると、どのような動作命令であっても即停止します。

#### ▶高速リミット点 $L_3$ , $L_4$

$L_3$ ,  $L_4$ は $L_1$ ,  $L_2$ の位置から減速ステップ数以上手前に設定します。キャリアは $L_3$ ,  $L_4$ に達すると初期設定データ(加減速パルス数)にしたがって減速停止します。

#### ▶基準点CNP

システムの運転開始時や、モータが脱調して現在位置がわからないとき、“基準点まで定速移動”命令で復帰することができます。

### 4-4 CPUとのインターフェース

本ボードでは、PPMC101CをCPUのI/Oポートとして扱っています。

I/Oアドレスは、上位8ビットをディップ・スイッチDIP<sub>1</sub>で、中位4ビットをDIP<sub>2</sub>で割り付け、下位4ビットのデコード出力をJP<sub>4</sub>(チャンネル1), JP<sub>5</sub>(チャンネル2)で選択します。

なお、JP<sub>3</sub>はPC9801Fまでの旧機種をBASICのみで使用する時、I/Oアドレスが下位8ビットしか有効でないため、B側に設定して上位8ビットを無視するためのジャンパ線です。

しかも、I/Oアドレスには空きが少なく、中位4ビットは通常Hex“D”しか使用できません。

PC9801M以降の機種では、BASICでも上位8ビットをHex“00”～“7F”まで使用できるので、JP<sub>3</sub>はW側に設定し、Hex 4桁のI/Oアドレスを使用するとよいでしょう。

なお、機械語プログラム中では、新旧に関係なくHex 4桁のI/Oアドレスを使用することができます(表4-1参照)。

さて、こうしてPPMC101CをCPU側からみると、2出力ポート(書き込みレジスタ)、2入力ポート(読み出しレジスタ)となります。

#### (1) 初期設定

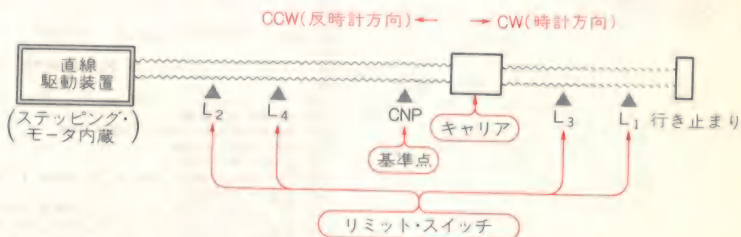
モータの種類や励磁方式、加減速データなどを設定します。

〈表4-1〉I/Oアドレス割り付け表

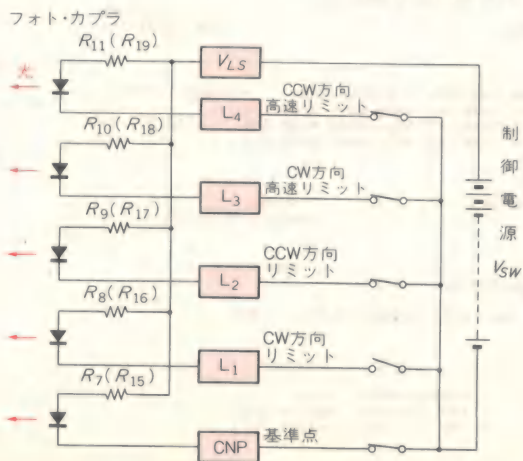
ジャンパ設定 JP <sub>4</sub> (チャンネル1) JP <sub>5</sub> (チャンネル2)	I/Oアドレス(Hex表示)	
	書き込み	読み出し
0	データ・レジスタ ××D0	コマンド・レジスタ ××D2
4	データ・レジスタ ××D4	ステータス・レジスタ ××D6
8	データ・レジスタ ××D8	ステータス・レジスタ ××DA
C	データ・レジスタ ××DC	ステータス・レジスタ ××DE

\* DIPスイッチの各ビットはONが“0”, OFFが“1”に対応する。  
\* “××”はDIP<sub>1</sub>による上位8ビット・アドレス(AB151-081)設定値Hexの00～7F

〈図4-4〉制御系モデル



〈図4-5〉リミット・スイッチ、基準点スイッチの接続例



#### (a) $R_7 \sim 11$ , および $R_{15} \sim 19$ の計算

各スイッチONのとき、入力フォト・カプラに10(8～12)mAの電流 $I_f$ が流れるような値とする。制御電源電圧を $V_{sw}$ 、フォト・カプラ入力発光ダイオードの電圧降下を約1Vとすると、

$$V_{sw} = 5V, I_f = 10mA \text{ ならば } R = (V_{sw} - 1) / I_f \\ = (5 - 1) / 0.01 \\ = 400\Omega$$

#### (b) リミット、基準点スイッチ素子

マイクロスイッチ、フォト・インタラプタなどを使用する。各接点はアクティブOFF(通常ONで動作時OFF)の構造のものを用いる。システムによっては、これら制御指標の一部、または全部を省略することもあるが、そのときは、省略したスイッチ部分を短絡した回路となるように接続する。

#### (c) 制御電源

小型ステッピング・モータを使用する場合は、モータ電源と共用してもよい。



## (2) 動作命令

具体的な動作命令(8種類ある), パルス数などのデータを書き込みます。この直後500 $\mu$ s以内に命令の実行が始まります。

## (3) レジスタ読み込み

動作終了後, 入出力端子の状態, 動作終了の原因, 残りパルス数などを読み込むことができます。

以上の操作命令一覧表を表4-2に示します。各命令の詳細についてはLSIのマニュアルを参照してください。

## 4-5 操作プログラム例

リスト4-1にBASICのみで記述したプログラム例を示します。各行の意味は行末にコメントを付けてありますから, 表4-2およびLSIのマニュアルと併読してください。

リスト4-2に実用的なプログラム例を示します。このプログラムでは, LSIの制御マシン・コードをBASICからCALL文で簡単に呼べる形の上位コマンド(1000~1290行)を用意し, ユーザは自身のメイ

ン・フローの中でこれをコールするだけでステッピング・モータを制御できるものです。

各コマンドの最終文字“n”はモータのチャンネル番号を, 引数(variable:変数)の中ほどにあるH,M,Lは当引数がデータの上位バイト, 中位バイト, 下位バイトであることを表しています。

またWはワード(16ビット/2バイト)データであることを意味し, 例えば“H MW”は上位バイト+中位バイトからなるワード・データを表します。

コマンド“SE n”, “SD n”, “FC”には引数がなく, 特に“FC”はLSIが動作コマンド実行終了後に要求する割り込みサービス・ルーチンでもあります。3000行以降がメイン・フローのプログラムです。

ほとんどの操作は, 引数にパラメータを代入した後, CALL文で動作コマンドを呼ぶ形になっています。


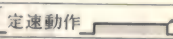

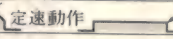

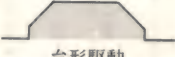



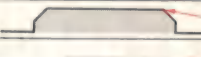
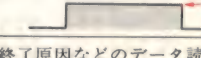
### ●参考・引用文献●

- (1)\*PPMC-101Cマニュアル, アンペール例。
- (2) デジタルIC活用完璧マスタ, §7シフトレジスタ, トランジスタ技術, 1986年5月号, p. 390。

〈リスト4-1〉  
BASICで記述したコントロール・プログラム

```
20 '
30 '
40 ' ...[ DIP2 を "D", JP3 を B (A*イット・モータ), JP4 を 0 にする ]...
50 '
60 CLS : CONSOLE 3,22,0 : PRINT " 「E」 テ ス ト プ ロ グ ラ ム " : PRINT : PRINT
70 '
80 ' ..... PULSE MOTOR driving JOB .....
90 '
100 ' ----- 初期設定 -----
105 GOSUB *CHECK
110 IF BUSY OR IBF=1 THEN 100
120 OUT &HD2,&H3F ' * 初期設定コマンド *
130 GOSUB *CHECK
135 IF IBF=0 THEN OUT &HD0,&HFF ELSE 130 ' RA max { 起動時のパルス・レート }
140 GOSUB *CHECK
145 IF IBF=0 THEN OUT &HD0,&HCO ELSE 140 ' RA min { 高速度時のパルス・レート }
150 GOSUB *CHECK:IF IBF=0 THEN OUT &HD0,&H0 ELSE 150 ' { 加減速の下位 }
160 GOSUB *CHECK:IF IBF=0 THEN OUT &HD0,&H8 ELSE 160 ' { パルス数(上位) }
170 '
180 FOR I=1 TO 20
190 GOSUB *CHECK : IF BUSY OR IBF=1 THEN 190
200 OUT &HD2,&H52 ' * シンク・スラップ *
205 FOR J=1 TO 500 : NEXT J ' [CW で 20ステップ]
210 NEXT I
220 '
230 GOSUB *CHECK
240 IF BUSY OR IBF=1 THEN 230 ELSE OUT &HD2,&H5B ' * 加減速動作 [CCW]
250 GOSUB *CHECK:IF IBF=0 THEN OUT &HD0,&H0 ELSE 250 ' { 下位 }
260 GOSUB *CHECK:IF IBF=0 THEN OUT &HD0,&H20 ELSE 260 ' { * パルス数 中位 }
270 GOSUB *CHECK:IF IBF=0 THEN OUT &HD0,&H0 ELSE 270 ' { 上位 }
280 '
290 '
300 GOSUB *CHECK
310 IF BUSY OR IBF=1 THEN 300 ' 動作終了待ち
320 OUT &HD2,&H80 ' * 終了ステータス読み込みコマンド *
330 GOSUB *CHECK
340 IF OBF<>1 THEN 330
350 EDATA=INP(&HD0) ' EDATA = 終了ステータス
360 PRINT "EDATA = ";HEX$(EDATA)
370 '
380 IF INKEY$<>"E" THEN 100 ELSE CONSOLE 0,25,1 : END
390 '
400 '
500 *CHECK
510 F=INP(&HD2)
520 BUSY=(F AND &H4)/4 ' BUSY=1:motor busy
530 IBF =(F AND &H2)/2 ' IBF =1:input buffer full
540 OBF = F AND &H1 ' OBF =1:output buffer full
550 RETURN
```

〈表4-2〉  
ステッピング・モータの  
コントロール用LSIの命令  
一覧表

		コマンド・データ		機 能
初期設定		1	0 0	
		2	(起動時パルス・レート)	
		3	(高速時パルス・レート)	
		4	〔加減速時パルス数〕	
		5		
動作命令	即 停 止	1	0 1 0 0 0	加減速動作  定速動作 
	減速停止	1	0 1 0 0 1	加減速動作  定速動作 
	シングルステップ	1	0 1 0 1 0	 1パルス
	加減速動作	1	0 1 0 1 1	
		2	〔動作ステップ数〕	
		3		台形駆動 
		4		三角駆動  (動作ステップ数が少ないとき)
	定速動作	1	0 1 1 0 0	
		2	(定速パルス・レート)	
		3	〔動作ステップ数〕	
リミット読み込み	リミットまで定速動作	1	0 1 1 0 1	 リミット・スイッチ L <sub>1</sub> or L <sub>2</sub> 点
	減速リミットまで動作	1	0 1 1 1 0	 リミット・スイッチ L <sub>1</sub> or L <sub>2</sub> 点
	基準点まで定速動作	1	0 1 1 1 1	 基準点CNP点
		2	(定速パルス・レート)	
レジスタ読み込み	終了データ	1	1 0 0 0 0 0 0 0	終了原因などのデータ読み込み 1バイト
	入力信号	1	1 0 0 0 0 0 0 1	リミット・スイッチなどのデータ読み込み1バイト
	出力信号	1	1 0 0 0 0 0 1 0	モータ相出力や方向データの読み込み 1バイト
	残りステップ数	1	1 0 0 0 0 0 1 1	残ったステップ数の読み込み 3バイト

# アナログ情報の宝庫

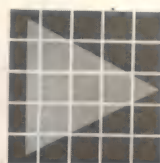
IC/LSI の情報は、回路設計者にとってはまさに欠かせないツールの一つです。ことに、一般には嫌われやすいアナログ回路ともなると、正しい特性の情報、わかりやすく役立つ応用技術情報がシステム設計上での有効な役割を果たします。そのような意味で、右に示すアナログ・デバイス社、パー・ブラウン社のデータ・ブックはアナログ回路(システム)、特に計測回路に関する面ではバイブルともいえる充実した内容になっています。

アナログ・デバイス社、パー・ブラウン社は、共に計測分野のアナログ回路をモジュール化、IC/LSI 化してきた世界のリーダ的なメーカーです。高精度の OP アンプはもとより、各種のアナログ演算回路、A-D/D-A コンバータ、データ収集システム、V-F/F-V コンバータ、アナログ・スイッチ/マルチプレクサなど、およそアナログ信号の計測、処理に必要なデバイスやボードがラインアップされています。あなたも一度、アナログの世界をのぞいてみませんか。

データ・ブックはもちろん和文です。

Databook の取り扱いは **CQ出版社**

アナログ・デバイス  
データブック  
DATA-ACQUISITION



ANALOG  
DEVICES

B5判 1796頁  
定 価 3000円  
送 料 450円

PRODUCTS  
アナログ データブック  
DATA BOOK



B5判 1390頁  
定 価 3600円  
送 料 400円



## 04

トランジスタ技術  
SPECIAL

Year	Model	Machine code	Subroutine	Tuning data
2000	FC	motor BUSY	7370777777	
2001	DATA 1K	50, 53, 52, 86, 00, 88, 00, 1E, 8E, C0, 26, 8E, 1E, F0,		
2002	DATA 00, B2	82, 8C, 24, 04, D0, 8E, 8E, C4, B2, D6, 8E, 24,		
2003	DATA 04, D0	88, 00, C4, 26, 8E, 1E, F0, 00, 8A, 07, 38, C4, 75, 05,		
2004	DATA B0, 80	8F, 00, 88, 27, 28, 0E, 88, C4, 80, 00, F8, FE, C0,		
2005	DATA D0, DC	73, FA, 88, 47, 08, 20, 8E, 08, 86, 00, 5A, 5B, 58,		
2006	DATA 07, 1F	CF		
2007	DATA 07, 1F	CF		
2008	DATA 07, 1F	CF		
2009	DATA 07, 1F	CF		
2010	DATA 50, 52, 82, 82, EC, 24, 02, 75, FA, 58, C3, 90, 90, 90, 90,			
2011	DATA 50, 52, 82, 82, EC, 24, 01, 74, FB, 5A, 58, C3, 90, 90, 90, 90,			
2012	DATA 50, 52, 82, 82, EC, 24, 06, 75, FA, 58, C3, 90, 90, 90, 90,			
2013	DATA 06, 56, 50, 53, 52, 86, 00, 88, 00, 0E, 8E, C1, 8B, 77, 0C,			
2014	DATA 26, 8A, 04, 8E, D4, FF, B2, D2, EE, 8F, 0A, 8E, C1, 8B, 77,			
2015	DATA 08, 26, 8A, 04, E8, A9, FF, B2, D2, EE, 8F, 0A, 8E, C1, 8B,			
2016	DATA 77, 04, 26, 8A, 04, E8, 98, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2017	DATA 87, 04, 26, 8A, 04, E8, 88, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2018	DATA B2, 0E, 5A, 59, 53, 52, 86, 00, 88, 00, 0E, 8E, C1, 8B, 77,			
2019	DATA B2, 0E, 5A, 59, 53, 52, 86, 00, 88, 00, 0E, 8E, C1, 8B, 77,			
2020	DATA 08, 26, 8A, 04, 44, E8, 78, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2021	DATA 26, 8A, 04, 44, E8, 78, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2022	DATA 88, 07, 26, 8A, 04, E8, 47, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2023	DATA C1, 8B, 77, 04, 26, 8A, 04, E8, 36, FF, B2, D2, EE, 8F, 0A, 8E,			
2024	DATA 8E, C1, 8B, 77, 04, 26, 8A, 04, E8, 26, FF, B2, D2, EE, 8F, 0A, 8E,			
2025	DATA 1E, FF, B2, D2, EE, 8F, 0A, 8E, 26, FF, B2, D2, EE, 8F, 0A, 8E,			
2026	DATA 06, 56, 50, 53, 52, 86, 00, 88, 00, 0E, 8E, C1, 8B, 77, 08,			
2027	DATA 26, 8A, 04, 44, E8, 78, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2028	DATA 88, 07, 26, 8A, 04, E8, 47, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2029	DATA C1, 8B, 77, 04, 26, 8A, 04, E8, 36, FF, B2, D2, EE, 8F, 0A, 8E,			
2030	DATA 8E, C1, 8B, 77, 04, 26, 8A, 04, E8, 26, FF, B2, D2, EE, 8F, 0A, 8E,			
2031	DATA 26, 8A, 04, 44, E8, 78, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2032	DATA 88, 07, 26, 8A, 04, E8, 47, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2033	DATA 1E, FF, B2, D2, EE, 8F, 0A, 8E, 26, FF, B2, D2, EE, 8F, 0A, 8E,			
2034	DATA 06, 56, 50, 53, 52, 86, 00, 88, 00, 0E, 8E, C1, 8B, 77, 08,			
2035	DATA 26, 8A, 04, 44, E8, 78, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2036	DATA 88, 07, 26, 8A, 04, E8, 47, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2037	DATA C1, 8B, 77, 04, 26, 8A, 04, E8, 36, FF, B2, D2, EE, 8F, 0A, 8E,			
2038	DATA 8E, C1, 8B, 77, 04, 26, 8A, 04, E8, 26, FF, B2, D2, EE, 8F, 0A, 8E,			
2039	DATA 26, 8A, 04, 44, E8, 78, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2040	DATA 88, 07, 26, 8A, 04, E8, 47, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2041	DATA 1E, FF, B2, D2, EE, 8F, 0A, 8E, 26, FF, B2, D2, EE, 8F, 0A, 8E,			
2042	DATA 06, 56, 50, 53, 52, 86, 00, 88, 00, 0E, 8E, C1, 8B, 77, 08,			
2043	DATA 26, 8A, 04, 44, E8, 78, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2044	DATA 88, 07, 26, 8A, 04, E8, 47, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2045	DATA C1, 8B, 77, 04, 26, 8A, 04, E8, 36, FF, B2, D2, EE, 8F, 0A, 8E,			
2046	DATA 8E, C1, 8B, 77, 04, 26, 8A, 04, E8, 26, FF, B2, D2, EE, 8F, 0A, 8E,			
2047	DATA 26, 8A, 04, 44, E8, 78, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2048	DATA 88, 07, 26, 8A, 04, E8, 47, FF, B2, D2, EE, 8F, 0A, 8E, C1,			
2049	DATA 1E, FF, B2, D2, EE, 8F, 0A, 8E, 26			



## PC9801用グレードアップ・インターフェース・ボードの作り方

本章では、PC9801をグレードアップするためのマウス・インターフェース・ボード、音声処理用ボード、ロジック・アナライザなどの作り方を詳細に解説します。

### 1 マウス・インターフェース・ボードの製作

最近のパソコンには、ほとんどといってよいほどマウスのインターフェースが標準装備されるようになってきました。筆者のPC9801F2は残念ながらマウス・インターフェースをもっていません。

しかし、マウス・インターフェース用のICが発売されていますから、自分でマウスのインターフェースを作ることが可能です。そこで、PC9801シリーズ用マウス・インターフェースの作り方と、マウスを使うためのBIOSの紹介をします。

インターフェースのタイミングなどで難しい点はありませんので、他のパソコンで使うことも簡単にできると思います。

#### 1-1 マウスについて

マイコンの入力装置には様々なものがあります。その中で、図やグラフ、座標を入力したり、スクリーン上で直接値を入力したりするものには、タブレットや

マウスがあります。

タブレットは絶対アドレスを入力する装置で、原点からの距離(オフセット)：絶対座標を入力できます。したがって、タブレットから図などを入力したりするのに適しています。

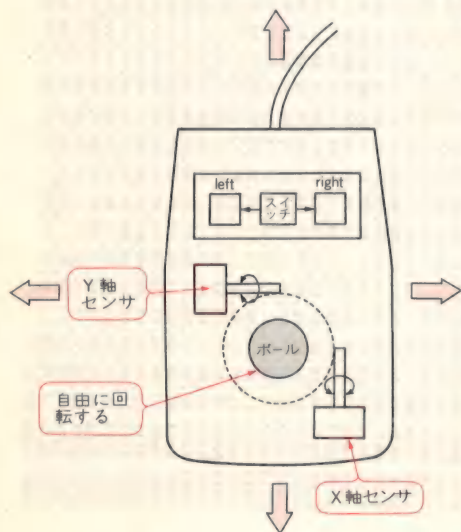
その点マウスは、今の場所からの移動量が入力されます。入力も今の場所からの距離：相対座標になります。

したがって、マウスの使用方法も図などの入力よりも、画面上でのグラフや図の作成、メニュー・モードのセレクトなどの画面上で行う作業の入力装置に適しています。

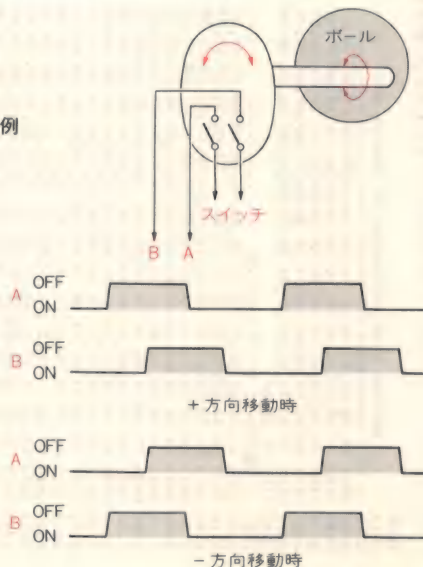
マウスは、本体を移動させることにより、その移動量を得ます。そのため、本体を移動できるように、底にボールが付いています。そのボールに、X軸とY軸の移動量を得るためのセンサが付いています(図1-1)。

実際には、センサは二つのスイッチから構成されていて、ボールをころがすことにより、位相差の異なる信号を出力します。その信号の位相差により、+の方向か-の方向に移動していることを示し、そのパルス

〈図1-1〉 マウスの構造



〈図1-2〉 マウス信号出力例



数で移動量を表します。図1-2にその信号の出力例を示します。

### 1-2 マウス・コントローラ $\mu$ PD4701ACについて

マウス用コントローラの $\mu$ PD4701ACの特徴を次に示します。

- ▶ X, Y 2 軸インクリメンタル方式ロータリ・エンコーダ用カウンタ
- ▶ カウント入力(シュミット・トリガ入力) 4 通倍カウント方式
- ▶ 12ビット・カウンタ(リセット値=000h)
- ▶ 8ビットTTLコンパチブル出力
- ▶ 3個分のキー入力バッファ
- ▶ C-MOS, +5V単一電源, 消費電力約500mW
- ▶ 24ピン・プラスチックDIP

図1-3にICの端子接続図を、図1-4に内部ブロック図を示します。また、表1-1に、各端子の機能を示します。

次に簡単に動作を示します。先ほど示したように、**二つの信号の位相差により+方向への移動か-方向への移動かを判断**します。このときのカウンタ動作を図1-5に示します。

このICには、カウンタ値が変化したときに、そのことを外部に知らせるフラグ(カウント・フラグ:CF)があります。このフラグにより、CPUに割り込みを

かけることができますが、通常の使用法では変化が多すぎて得策だとはいえません。ポーリングなどでステータスとして取り込むのに便利です。

さらに、このICには**スイッチ用のバッファが三つあります**。マウスに付いているスイッチをそのまま取り込むことができます。これにも、CFと同じように、フラグ(スイッチ・フラグ:SF)があります。どれか一つでもスイッチがONのとき、SFが“L”になります。

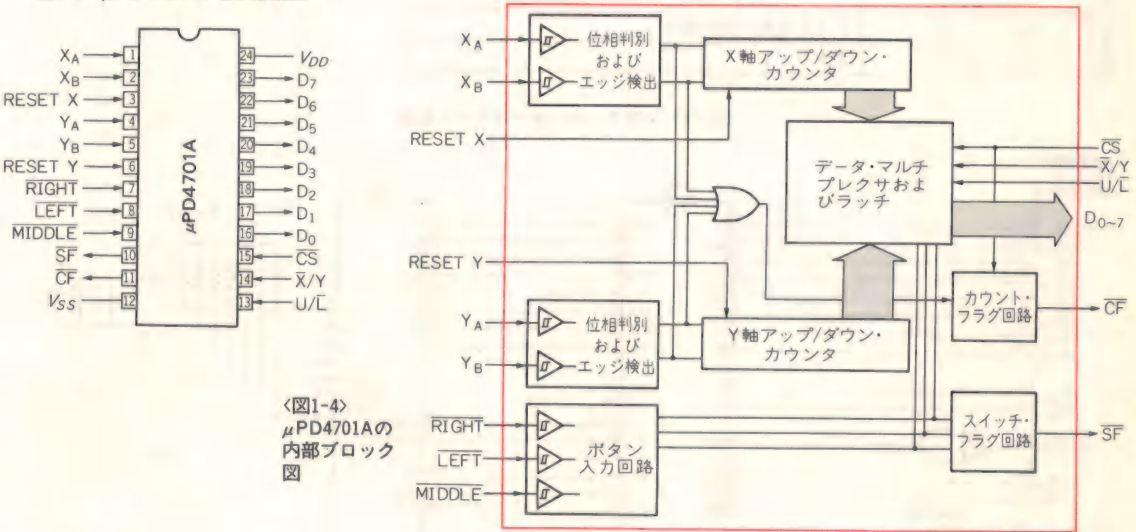
このフラグも、CPUの割り込みに入れることができますが、**どれか一つでもONのときに立つので、一つがONで後で他のスイッチがONになってもSFは変化しません**。したがって、このフラグもステータスとして見るほうが便利です。

### 1-3 PC9801用マウス・インターフェース・ボード

PC9801の拡張I/Oとしてボードを作る場合、最初にI/Oアドレスを決めなければなりません。PC9801では、ユーザ用にD0h~DFhが開放されていますが、VMなどではマウス・インターフェース用にD1hを使っています。したがって、ここでは安全のために、**D8hから使用**することにします。

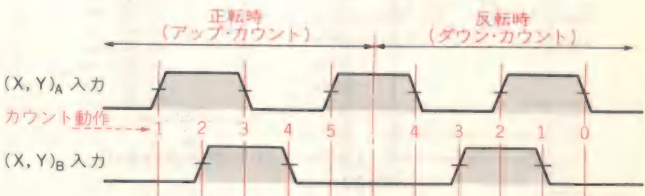
$\mu$ PD4701AはCPUに直接接続できるようになっています。しかし、カウンタの値はCSでラッチされるようになっており、すべてのカウンタ値を読み出すためには、4回のアクセス(X方向2回,Y方向2回)が

〈図1-3〉  $\mu$ PD4701Aの端子接続図



〈図1-4〉  
 $\mu$ PD4701Aの  
内部ブロック  
図

〈図1-5〉  $\mu$ PD4701Aのカウンタ動作





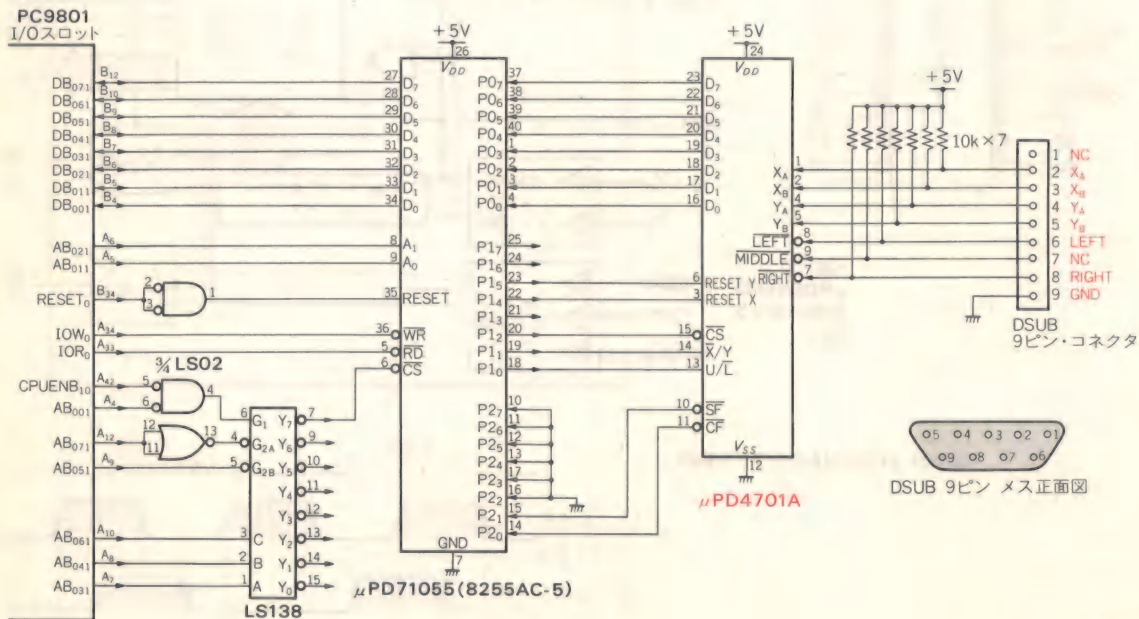
〈表1-1〉  $\mu$ PD4701Aの端子機能

	端子名	入出力	機能							
CPUとのインターフェース部	$\overline{CS}$	入力	チップ・セレクト入力。"L"入力ではD <sub>0</sub> ~7出力をアクティブにする。"H"入力では、D <sub>0</sub> ~7出力は、ハイ・インピーダンスとなる。また、 $\overline{CS}$ の立ち下がりエッジで出力データがラッチされる							
	$\overline{X}/Y$	入力	カウンタ・セレクト入力。"L"入力ではXカウンタを、"H"入力ではYカウンタを選択する							
	$U/\overline{L}$	入力	バイト・セレクト入力。"L"入力では下位のバイトを、"H"入力では上位のバイトを選択し、データの出力をコントロールする							
	RESET X RESET Y	入力	カウンタのリセット入力。RESET X入力によりXカウンタが、RESET Y入力によりYカウンタがリセットされる。いずれも"H"アクティブ							
	D <sub>0</sub> ~7	出力 (3ステート)	CPUへのデータ出力バス、 $\overline{X}/Y$ および $U/\overline{L}$ 入力により選択されたバイト・データが出力される。データは $\overline{CS}$ の立ち下がりエッジでラッチされたものが出力される。							
	$\overline{CF}$	出力	カウント・フラグ出力。 $\overline{CS}$ ="H"の期間中、XまたはYカウンタが変化したときセット(="L"出力)される。 $\overline{CS}$ の立ち下がりエッジでリセット(="H"出力)され、 $\overline{CS}$ ="L"の期間中は、カウント・フラグの出力は、禁止され、"H"レベルが出力される							
	$\overline{SF}$	出力	スイッチ・フラグ出力。スイッチ入力RIGHT, LEFT, MIDDLEのいずれかが"L"の期間中、アクティブ(="L"出力)となる							
マウスとのインターフェース部	X <sub>A</sub> , X <sub>B</sub>	入力 (シュミット入力)	Xカウンタ用2相信号入力端子							
	Y <sub>A</sub> , Y <sub>B</sub>	入力 (シュミット入力)	Yカウンタ用2相信号入力端子							
	$\overline{RIGHT}$ $\overline{LEFT}$ $\overline{MIDDLE}$	入力 (シュミット入力)	キー・スイッチ入力端子。キー・スイッチ入力は、内部ステータスとして、XカウンタおよびYカウンタの上位バイトの上位4ビットとして、読み出される。 上位バイト <table border="1"><tr><td>SF</td><td>L</td><td>R</td><td>M</td><td>C<sub>11</sub></td><td>C<sub>10</sub></td><td>C<sub>9</sub></td><td>C<sub>8</sub></td></tr></table> キー入力ステータスカウンタ・データ	SF	L	R	M	C <sub>11</sub>	C <sub>10</sub>	C <sub>9</sub>
SF	L	R	M	C <sub>11</sub>	C <sub>10</sub>	C <sub>9</sub>	C <sub>8</sub>			
電源部	V <sub>DD</sub>		+ 5 V 電源の接続端子							
	V <sub>SS</sub>		グラウンド							

〈表1-2〉 I/Oマップ

アドレス		
00D8	ポート 0	$\mu$ PD 71055C
00DA	ポート 1	
00DC	ポート 2	
00DE	コントロール	

〈図1-6〉 マウス・インターフェース回路



必要になります。

この読み出している間にカウンタ値が変化したときには、 $\overline{\text{CS}}$ を一度インアクティブにすると、新しい値がカウンタにラッチされ、正しいデータとして連続して読み出すことができません。したがって、**カウンタ値を読み出すときは、 $\overline{\text{CS}}$ をアクティブのままにしておいたほうがよいでしょう。**

ここでは、ハードを簡単にするために、 $\mu\text{PD71055}$  Cを使用しました。 $\mu\text{PD71055C}$ は8255AをC-MOSに

したもので、ピン・コンパチブルで、消費電力が小さいという特徴があります。リセット信号も、この $\mu\text{PD71055C}$ から作り出しています。

図1-6に全回路図を示します。表1-2にI/Oマップを示します。また、 $\mu\text{PD4701A}$ の特性を表1-3と図1-7に示します。

まず、 $\mu\text{PD71055C}$ を初期化します。モードを0としポート0を入力、ポート1を出力、ポート2を入力と設定します。 $\mu\text{PD71055C}$ はC-MOSなので、入力

〈表1-3〉  $\mu\text{PD4701A}$ の電気的特性

●絶対最大定格 ( $T_a = 25^\circ\text{C}$ ,  $V_{SS} = 0\text{V}$ )

項目	略号	定格値	単位
電源電圧	$V_{DD}$	$-0.5 \sim +7.0$	V
入力電圧	$V_I$	$-1.0 \sim V_{DD} + 1.0$	V
出力電圧	$V_O$	$-0.5 \sim V_{DD} + 0.5$	V
動作温度	$T_{op}$	$-40 \sim +85$	$^\circ\text{C}$
保存温度	$T_{stg}$	$-65 \sim +150$	$^\circ\text{C}$
許容損失	$P_D$	500	mW

●DC特性

( $T_a = -40 \sim +85^\circ\text{C}$ ,  $V_{DD} = +5\text{V} \pm 10\%$ )

項目	略号	条件	規格値		単位
			min	max	
"L" レベル入力電圧	$V_{IL}$			0.8	V
"H" レベル入力電圧	$V_{IH}$	$X_A, X_B, Y_A, Y_B$ および LEFT, RIGHT, MIDDLE	2.6		V
	$V_{IH}$	上記以外	2.2		V
"L" レベル出力電圧	$V_{OL}$	$I_{OL} = 12\text{mA}$		0.45	V
"H" レベル出力電圧	$V_{OH}$	$I_{OH} = -4\text{mA}$	$V_{DD} - 0.8$		V
静消費電流	$I_{DD}$	$V_I = V_{DD}, V_{SS}$		50	$\mu\text{A}$
入力電流	$I_I$	$V_I = V_{DD}, V_{SS}$	-1.0	1.0	$\mu\text{A}$
3ステート出力リーク電流	$I_{OFF}$		-10	10	$\mu\text{A}$
動消費電流	$I_{DD\text{ dyn}}$	$f_{IN} = 500\text{kHz}$		2	mA
ヒステリシス電圧		$X_A, X_B, Y_A, Y_B$ および LEFT, RIGHT, MIDDLE	0.25		V

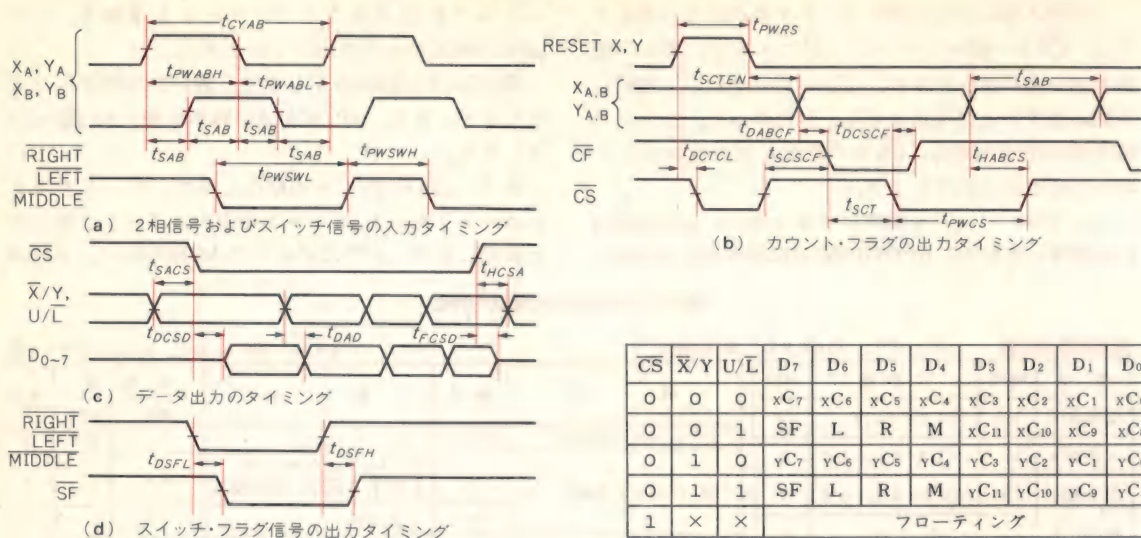
●AC特性

( $T_a = -40 \sim +85^\circ\text{C}$ ,  $V_{DD} = 5\text{V} \pm 10\%$ )

項目	略号	条件	規格値		単位
			min	max	
$X_A, B$ $Y_A, B$	入力周期	$f_{CYAB}$ $f_{IN} = 500\text{kHz}$	2		$\mu\text{s}$
	"H" レベル・パルス幅	$t_{PWABH}$	900		ns
	"L" レベル・パルス幅	$t_{PWABL}$	900		ns
	信号位相差時間	$t_{SAB}$	350		ns
$\overline{R}, \overline{L}$ $\overline{M}$	"H" レベル・パルス幅	$t_{PWSWH}$ スイッチ OFF時	30		$\mu\text{s}$
	"L" レベル・パルス幅	$t_{PWSWL}$ スイッチ ON時	30		$\mu\text{s}$
$\overline{\text{SF}}$	セット遅延時間	$t_{DSFL}$ スイッチ ON時		50	ns
	リセット遅延時間	$t_{DSFH}$ スイッチ OFF時		50	ns
RESET $X, Y$	パルス幅	$t_{PWRS}$	100		ns
	カウント・イネーブル時間	$t_{SCTEN}$ 対RESET $X, Y \downarrow$	0		ns
	カウンタ・クリア時間	$t_{DCTCL}$ 対RESET $X, Y \uparrow$		100	ns
$\overline{\text{CF}}$	フラグ・セット時間	$t_{DABCF}$ 対 $X_A, B, Y_A, B$		120	ns
	フラグ・リセット時間	$t_{DCSCF}$ 対 $\overline{\text{CS}} \downarrow$		100	ns
	カウンタ・セット時間	$t_{SCT}$ 対 $\overline{\text{CF}} \downarrow$	0		ns
$\overline{\text{CS}}$	$\overline{\text{CF}}$ イネーブル時間	$t_{SCSCF}$ 対 $\overline{\text{CF}} \downarrow$		140	ns
	$\overline{\text{CF}}$ ディセーブル時間	$t_{HABCS}$ 対 $X_A, B, Y_A, B$		100	ns
	パルス幅	$t_{PWCS}$		200	ns
$\overline{X}/Y$ $U/\overline{L}$	アドレス設定時間	$t_{SACS}$ 対 $\overline{\text{CS}} \downarrow$	0		ns
	アドレス保持時間	$t_{HCSAB}$ 対 $\overline{\text{CS}} \uparrow$	0		ns
$D_0 \sim 7$	出力遅延時間	$t_{DCSD}$ 対 $\overline{\text{CS}} \downarrow$		150	ns
	出力遅延時間	$t_{DAD}$ 対 $\overline{X}/Y, U/\overline{L}$		100	ns
	フローティング時間	$t_{FCSd}$ 対 $\overline{\text{CS}} \uparrow$		50	ns



〈図1-7〉  $\mu$ PD4701Aの電気的特性



〈リスト1-1〉 初期化と読み取りプログラム部を含むBIOS

```

PIO0 EQU 00D8H ;PIO port 0 アドレス
PIO1 EQU 00DAH ;PIO port 1 アドレス
PIO2 EQU 00DCH ;PIO port 2 アドレス
PIOC EQU 00DEH ;PIO コントロール・ポート

PUBLIC MOUSE_INIT ;Mouse BIOS 初期化

CODE SEGMENT PUBLIC
ASSUME CS:CODE

;=====
; BIOS initialization
;=====
MOUSE_INIT:
    PUSH AX ;レジスタのセーブ
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH ES

    MOV DX,PIOC ;PIO 初期化
    MOV AL,10010101B ; port 0 = 入力
    OUT DX,AL ; port 1 = 出力
    MOV DX,PIO1 ; port 2 = 入力
    MOV AL,00110111B ;μPD4701Aのリセット
    OUT DX,AL

    MOV AX,0000H ;MOUSE位置の初期化
    MOV CS:[X0_ADR],AX ; X アドレスの初期化
    MOV CS:[X1_ADR],AX
    MOV CS:[Y0_ADR],AX ; Y アドレスの初期化
    MOV CS:[Y1_ADR],AX
    MOV AL,00H ;スイッチのデータの初期化
    MOV CS:[SWITCH],AL ;すべてOFF

    MOV AX,0000H ;割り込みベクタの設定
    MOV ES,AX
    MOV DX,CS

    MOV AX,OFFSET MOUSE ;set BIOS entry point
    MOV ES:[40H*4],AX ;40H (BIOS call)
    MOV ES:[40H*4+2],DX ;set CS

    CALL MOUSE_TIME_SET ;インターバル・タイムのセット
    ; (in PC9801)

    POP ES ;レジスタの復元
    POP DX
    POP CX
    POP BX
    POP AX
    RET ;Normal return

;=====
; MOUSE BIOS エントリ・ポイント
;=====
MOUSE:
    MOV BX,CS:[X1_ADR] ; X アドレスを得る
    MOV CX,CS:[Y1_ADR] ; Y アドレスを得る
    MOV AL,CS:[SWITCH] ; スイッチのステータスを得る
    IRET

;=====
; Interval process
;=====
MOUSE_INT:
    PUSH AX ;レジスタのセーブ
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH DS

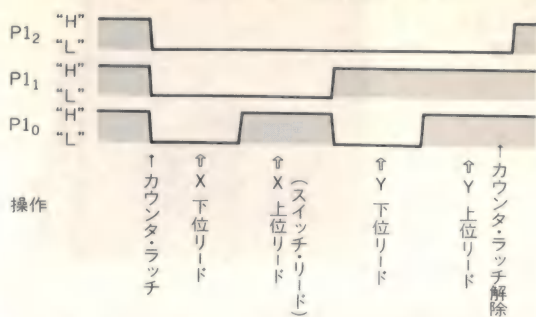
    MOV DX,PIO1 ;カウンタの読み出し
    MOV AL,00000000B ; CS=1, Xの低位側アドレス・リード
    OUT DX,AL

    MOV DX,PIO0 ; Xの低位側アドレス・リード
    IN AL,DX
    MOV BL,AL
    MOV DX,PIO1 ; Xの上位側アドレス・リード
    MOV AL,00000001B
    OUT DX,AL
    MOV DX,PIO0 ; リード
    IN AL,DX
    MOV BH,AL

    MOV DX,PIO1 ; Y アドレスのリード
    MOV AL,00000010B ; Yの低位側アドレス
    OUT DX,AL
    MOV DX,PIO0 ; リード
    IN AL,DX
    MOV CL,AL
    MOV DX,PIO1 ; Yの上位側アドレス
    MOV AL,00000011B
    OUT DX,AL
    MOV DX,PIO0 ; リード
    IN AL,DX

```

〈図1-8〉 カウンタ・リードのタイミング



使わない部分は+5V(プルアップ)かGNDの電位に固定しておきます。

ポート 0 は 8255A の  $PA_x$  に、ポート 1 は  $PB_x$ 、ポート 2 は  $PC_x$  に相当します。

次に、 $\mu$ PD4701Aをリセットします。RESET X、RESET Yを“H”にします。次に“L”にします。これでリセット完了です。リスト1-1のMOUSE IN

ITに、プログラムを示します。

次にデータの読み出し方について示します(今回は  $\overline{\text{SF}}$ ,  $\overline{\text{CF}}$  を使用していない). まず,  $\overline{\text{CS}}$  を “L” にし, カウンタ値をラッチします. このとき,  $\overline{\text{X}}/\text{Y}$ ,  $\text{U}/\overline{\text{L}}$  を 00 とし, X カウンタの下位を読みます. 次に 01 とし, X カウンタの上位とスイッチの状態を読みます.

同じように、Yカウントを読み出します。最後に  $\overline{\text{CS}}$  を“H”にします。

なお、カウント値を読み出したときのビット・マップを表1-4に示します。

マウスは今回、MICROSOFT製のPC9801用マウスを使用しました。

#### 1-4 BIOSについて

では、マウスを使うための簡単なBIOSをMS-DOS上で作ってみます。実際に使用するときは、アプリケーション・プログラムとBIOSをリンクすることにより、基本部分で細かい部分はBIOSをコールするだけ

MOV	CH,AL	:	
AND	AL,01110000B	:	スイッチのステータスのセット
MOV	CS:[SWITCH],AL	:	
PUSH	CX	:	
AND	BX,0FFFF	:	Xアドレスの訂正
MOV	AX,CS:[X1_ADDR]	:	
MOV	CS:[X0_ADDR],AX	:	
MOV	CX,640	:	Xのアドレスのリミット
CALL	ADR_CAL	:	
MOV	CS:[X1_ADDR],AX	:	
POP	CX	:	
AND	CX,0FFFF	:	Yアドレスの訂正
MOV	BX,CX	:	
MOV	AX,CS:[Y1_ADDR]	:	
MOV	CS:[Y0_ADDR],AX	:	
MOV	CX,400	:	Yアドレスのリミット
CALL	ADR_CAL	:	
MOV	CS:[Y1_ADDR],AX	:	
MOV	DX,PI01	:	カウンタの初期化
MOV	AL,CS:[0110111B]	:	
OUT	DX,AL	:	
MOV	AL,00000111B	:	
OUT	DX,AL	:	
MOV	AL,0	:	カラー = 0
MOV	BX,CS:[X0_ADDR]	:	マウスを消す
MOV	CX,CS:[Y0_ADDR]	:	
CALL	SET_MOUSE	:	
MOV	AL,7	:	カラー = 7
MOV	BX,CS:[X1_ADDR]	:	
MOV	CX,CS:[Y1_ADDR]	:	マウスのセット
CALL	SET_MOUSE	:	
CALL	MOUSE_TIME_SET	:	インターバル・タイマの再スタート
POP	DS	:	レジスタ復帰
POP	ED	:	
POP	DX	:	
POP	CX	:	
POP	BX	:	
POP	AX	:	
IRET		:	

R_CAL:	TEST	BX,0800H	; アドレスの計算
	JNZ	MINUS_MOVE	; + move or - move
	ADD	AX,BX	; + move
	CMP	AX,CX	; レンジを越えた?
	JNC	ADR_CAL_OVR	;
	RET		;
R_CAL_OVR:	MOV	AX,CX	; リミット値のセット
	DEC	AX	; -1
	RET		;

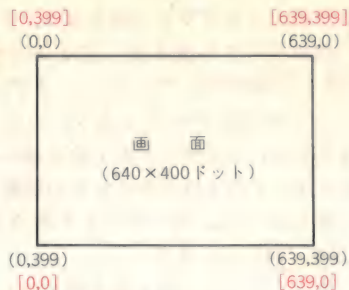
```

MINUS_MOVE:      NOT      BX          ; - move
                  AND      BX,0FFFFH  ;
                  INC      BX          ;
                  SUB      AX,BX       ; 2' complement
                  JC       ADR_CAL_LOW ; レンジを超えた
                  RET
;
ADR_CAL_LOW:     MOV      AX,0         ; 0のセット
                  RET
;
;
;
SFT_MOUSE:       MOV      CS:[COLOR],AL ; MOUSEへの書き込み
                  MOV      CS:[X_ADR],BX ; カラーのセット
                  MOV      CS:[Y_ADR],CX ; Xのアドレス
                  MOV      AX,0001      ; Yのアドレス
                  MOV      CS:[DOTS],AX ; ドット
                  MOV      AX,OFFSET BUF ; 作業領域
                  MOV      CS:[WBUF],AX ;
                  MOV      AX,00FFH     ; パターン
                  MOV      CS:[PATN],AX ;
                  ;
                  PUSH     CS           ;
                  PUSH     CS           ;
                  POP      DS           ;
                  POP      ES           ;
                  MOV      CH,10110000B ; CRTモード
                  MOV      BX,OFFSET GRAPH
                  MOV      AH,45H       ;
                  INT      18H          ;
                  RET
;
;
;
=====
Working area
=====
;
;
X0_ADR           DW      0             ; X old address
X1_ADR           DW      0             ; X new address
Y0_ADR           DW      0             ; Y old address
Y1_ADR           DW      0             ; Y new address
;
SWITCH           DB      0             ; SWITCH data
;
GRAPH:
COLOR            DB      0,0           ; color
                  DW      0,0,0
X_ADR            DW      0             ; X_address
Y_ADR            DW      0             ; Y_address
DOTS             DW      0             ; dots
WBUF             DW      0             ; working buffer address
                  DW      10H DUP(0)
;
BUFF:
PATN             DW      0             ; pattern
                  DW      40H DUP(0)
;
;
;
CODE            ENDS
;
;
;
END

```



〈図1-9〉  
マウス用BIOS内の  
アドレスと画面の  
アドレス対応



(x,y) 画面のアドレス

[X,Y] マウス用BIOS内のアドレス

で行うことができます。

BIOSとして装備する機能を以下のように決めます。ここでマウスのカウント値は、インターバル・タイマで読み出すものとします。また、ここでは、BIOSの作り方と使い方を示すだけで、実際に作る場合は、自分の使い方に合ったBIOSを作成してください。

- (1) マウスの現在位置
- (2) マウスの位置の表示
- (3) スイッチの状態
- (4) BIOSの初期化

処理を画面上だけで行うことを仮定し、マウスの移動ができる範囲を(0,0)～(639,399)とし、これ以上は移動できないとします。

MOUSE\_INITで、座標、PIO、割り込みベクタの初期化と、マウスの位置を読み出すためのタイマ・ルーチンを起動します。

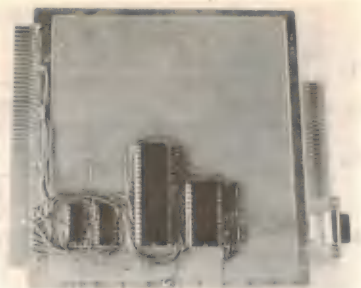
タイマは、PCのBIOSを使用しました。タイマ・ルーチンでは、100msごとにマウスの移動量を得て、その値を前の座標に加えることにより現在値を求めます。マイナスの移動に関しては、負の値を正の値に変換して引いています。また、範囲外に関しては、境界値をセットするようにしています。今回のBIOSは、エントリ・ポイントを1ポイント(ベクタ:40h)とし、マウスの現在場所を得るだけにしています。マウスの表示は、1ドットで表しています。

リスト1-2に示すアプリケーション・プログラムは、BIOSをコールし、スイッチのステータスとアドレスを得ます。左のスイッチを押すとスタート点を記憶します。次に、右のスイッチを押すと、スタート点から今の点まで直線を引きます。

また、画面クリアと、ラインを引くプログラムも載せておきます。

今回、μPD4701Aを使って、PC9800用マウス・インターフェースを作ってみました。μPD4701Aが非常に使いやすく、プログラムも容易に行うことができました。ただし、このままでは一般に売り出されているマウスを使ったソフトが使えませんが、PC9800Vシ

〈写真1-1〉  
表面から見た  
部品配置



リーズの内部構造(マウス関連のインターフェース)が明確にされれば、同じインターフェースにすることも可能です。

●参考・引用文献●

- (1) 日本電気㈱, μPD4701ACマニュアル.
- (2) 日本電気㈱, PC9801VMユーザーズマニュアル.

## 2 音声処理ボードの製作

### 2-1 音声処理ボードの仕様と概要

パソコンの拡張スロット用A-DおよびD-Aコンバータ・ボードが各種市販されており、種々のアプリケーションに利用されています。それらを組み合わせて、音声処理システムを構築しようとする、スペックの点から、

- ① A-D変換スピードは100μsで十分(市販品は25μsぐらいが多い)。
- ② A-Dコンバータの分解能は10ビットで十分(同じく12ビットが多い)。
- ③ 入力チャンネル数は、多くても2チャンネルで十分。

また、構成上の観点から、

- ① A-Dコンバータ・ボードとD-Aコンバータ・ボードの二つのボードを必要とし、スロットも二つ占有される。
- ② マイクロホン・アンプ、A-D変換の前置フィルタ(アンチ・エイリアシング・フィルタ)、D-A変換後のローパス・フィルタなど、自分で用意するものが結構たくさんある。
- ③ A-Dコンバータ・ボード用とD-Aコンバータ・ボード用に別々のコントロール・ソフトを必要とし、多くの場合、非常に操作が煩わしくなる。

などの点で不都合があり、パソコンを利用して手軽にシステムを組むという、拡張ボード本来のメリットが生きてきません。

市販A-D変換ボードの代表的スペックの変換スピード数十μs/ワード、分解能12ビットという値は、専用機と比べてもひけを取りません。しかし、それらの性能を十二分に引き出すには、パソコン拡張スロット

＜リスト1-2＞ CRT上でマウスを動かすためのアプリケーション・プログラム

```

EXTRN    MOUSE_INIT:      NEAR
;-----
CODE     SEGMENT PUBLIC   CS:CODE,DS:DATA,SS:SSEG ASSUME DS,SSEG
;-----
MOV      AX,SEG DATA
MOV      DS,AX
; ;データ・セグメントのセット
; ;
MOV      AX,SEG SSEG
MOV      SS,AX
; ;SSセグメントのセット
; ;
MOV      AX,OFFSET STACK_POINTER
MOV      SP,AX
; ;スタックの設定
; ;
MOV      AH,42H
MOV      CH,11100000B
INT      18H
; ;CRT モードのセット
; ; 640×400
; ;
MOV      AH,40H
INT      18H
; ;CRT ON
; ;
CALL     CLEAR_CRT
; ;CRT クリア
; ;グラフィック画面のクリア
CALL     MOUSE_INIT
; ;MOUSE BIOS初期化
; ;
STI
; ;
INT      40H
; ;Call BIOS (アドレスを得る)
; ;
TEST     AL,01000000B
JZ       CHECK_NEXT
; ;左のスイッチのチェック
; ;右のスイッチのチェック
; ;
MOV      [X1_ADR],BX
MOV      [Y1_ADR],CX
; ;X1アドレスのセーブ
; ;Y1アドレスのセーブ
; ;
CHECK_NEXT:
TEST     AL,00100000B
JZ       CHECK_END
; ;右のスイッチのチェック
; ;
MOV      [X2_ADR],BX
MOV      [Y2_ADR],CX
; ;X2アドレスのセーブ
; ;Y2アドレスのセーブ
; ;
MOV      AX,[X1_ADR]
MOV      BX,[Y1_ADR]
MOV      CX,[X2_ADR]
MOV      DX,[Y2_ADR]
CALL     WRITE_LINE
; ;線を描く
; ;(X1,Y1)-(X2,Y2)
; ;
CHECK_END:
MOV      AH,1
INT      18H
CMP      BH,01
JNZ      MAIN_LOOP
; ;キー・スキャン
; ; if KEY_IN then EXIT
; ;
MOV      AH,4CH
INT      21H
; ;Dummy return
; ; return to OS
; ;
=====
clear_crt : CRT initialization
subroutine
=====
clear_crt:
;clear CRT
push     ds
mov      ax,0a000h
mov      ds,ax
; ;スタート・アドレスのセット
; ;
push     bx
push     cx
call     clear_crt_a
call     clear_crt_c
; ;全スクリーンをクリア
; ;
pop      ox
pop      dx
pop      bx
ret
; ;
clear_crt_a:
mov      bx,0
mov      cx,80*25
mov      al,0
mov      ah,11100001b
; ;テキスト・スクリーンのクリア
; ;オフセットをゼロにする
; ;80文字×25行
; ;文字クリア
; ;アトリビュートのセット
; ;
clear_crt_com:
mov      [bx],al
mov      byte ptr [bx+1],00h
mov      mov      2000h [bx],ah
add      bx,2
loop     clear_crt_com
; ;アトリビュートのセット
; ;increment BX (pointer)
; ;
clear_crt_c:
mov      ax,0a800h
mov      ds,ax
mov      bx,0
mov      cx,40*400
call     clear_gort_com
; ;グラフィック・スクリーンのクリア
; ;スタート・アドレスのセット
; ;オフセット・アドレスのセット
; ;クリア画面数
; ;有画面のクリア
; ;
mov      ax,0b000h
mov      ds,ax
mov      bx,0
mov      cx,40*400
call     clear_gort_com
; ;スタート・アドレスのセット
; ;オフセット・アドレスのセット
; ;クリア画面数
; ;有画面のクリア
; ;
mov      ax,0b800h
mov      ds,ax
mov      bx,0
mov      cx,40*400
call     clear_gort_com
; ;スタート・アドレスのセット
; ;オフセット・アドレスのセット
; ;クリア画面数
; ;有画面のクリア
; ;
wait_gdc_2:
in       al,0a0h
test     al,04h
jz       clear_gort_com
; ;clear graphic screen common
; ;wait until FIFO become empty
; ;not empty FIFO
; ;
push     ax
push     ax
pop      ax
pop      ax
; ;dummy instruction
; ;
fill_space_gvram:
mov      [bx],ax
add      bx,2
loop     fill_space_gvram
; ;fill space
; ;
write_line:
entry    line (X1,Y1)-(X2,Y2)
; ;
; ; AX = X1
; ; BX = Y1
; ; CX = X2
; ; DX = Y2
; ;
push     ds
push     dx
push     cx
push     bx
push     ax
; ;X1のセーブ
; ;X2のセーブ
; ;Y1のセーブ
; ;X1のセーブ
; ;
push     ca
pop      ds
; ;
mov      si,offset graph_work
; ;white line
; ;
mov      al,07
mov      ds:[si+0h],al
; ;X1を得る
; ;X1のセーブ
; ;
mov      ds:[si+8h],ax
; ;X1のセーブ
; ;Y1のセーブ
; ;
mov      ds:[si+0ah],ax
; ;X2を得る
; ;X2のセーブ
; ;
mov      ds:[si+16h],ax
; ;X2のセーブ
; ;Y2のセーブ
; ;
mov      ds:[si+18h],ax
; ;Y2のセーブ
; ;
; ;
; ; ライン・パターンをセット
; ; "1111111111111111"
; ; set line
; ;
mov      ax,0ffffh
mov      ds:[si+20h],ax
; ;
mov      al,01
mov      ds:[si+28h],al
; ;CRT パラメータ
; ;オフセットのセット
; ;
mov      ch,10110000b
mov      bx,si
mov      ah,47h
int      18h
; ;call line(X1,Y1)-(X2,Y2)
; ;
pop      ds
; ;
graph_work: db 100 dup(0)
; ;
CODE     ENDS
;-----
DATA     SEGMENT PUBLIC   CS:CODE,DS:DATA,SS:SSEG ASSUME DS,SSEG
;-----
X1_ADR   DW 0
Y1_ADR   DW 0
X2_ADR   DW 0
Y2_ADR   DW 0
; ;X1アドレス
; ;Y1アドレス
; ;X2アドレス
; ;Y2アドレス
; ;
DATA     ENDS
;-----
SSEG     SEGMENT STACK   CS:CODE,DS:DATA,SS:SSEG ASSUME DS,SSEG
;-----
STACK_POINTER DW 100H DUP(0)
; ;スタック領域
; ;
SSEG     ENDS
;-----
END

```



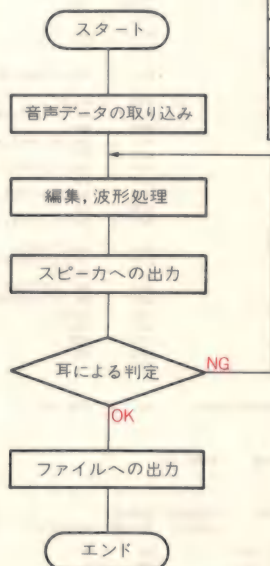
〈表2-1〉 音声処理ツール・ボードの仕様

入力チャネル	8チャネル (ch1~ch7:直結 ch8:マイクロホン入力)
ゲイン	100~500倍
アンチエイリアシング・フィルタ	カットオフ周波数 4kHz 24dB/oct パタワース特性
A-Dコンバータ	サンプリング速度 100 $\mu$ s 分解能 8ビット
D-Aコンバータ	分解能 8ビット
ローパス・フィルタ	カットオフ周波数 4kHz 12dB/oct パタワース特性
出力アンプ	最大電力 0.5W
プログラマブル・タイマ	1.25 $\mu$ s~97734hour
デジタル入出力	入力: 1ビットTTLレベル 出力: 1ビットTTLレベル
CPUインターフェース	上位4ビットがディップ・スイッチで設定可能なI/Oポート、転送方式は、ポーリング、インタラプト、DMAのいずれも可能
マイクロホン	エレクトレット・コンデンサ型マイクロホン
スピーカ	0.5W 8 $\Omega$

〈表2-2〉 コマンドの一覧表

A	acquisition	音声データの取り込み
B	between	カーソル間時刻/周波数表示
C	change cursor mode	カーソル・モードの変更
D	delete	データの指定部分の削除
E	exit	エディット終了、メインへ
F	FFT	フーリエ変換
G	get dir	ディレクトリ表示
H	hard copy	プリンタへのハード・コピー
I	insert	指定部への指定データの挿入
J	jump	フリー・カーソルのジャンプ・モード指定
K	kill	指定番号のタグ・カーソルの削除
L	local	プロセス領域の指定
M	move	指定部への表示窓の移動
N	new channel	チャネル・モードの変更
O	over write	指定部への指定データの重ね書き
P	print text	画面へのテキスト記入モード
Q	quit	エディット終了、OSへ
R	read	データ・ファイルの読み出し
S	show tags	タグ・カーソルの表示
T	tags set	タグ・カーソル設定
U	unite	データ・ファイルの結合
V	voice out	データの指定部分の音声出力
W	write	データ・ファイルの書き込み
X, Y	X-axis, Y-axis	X, Y軸の原点およびスケール設定
Z	zoom	X軸の縮小または拡大

〈図2-1〉 音声合成の開発フローチャート



内でのノイズ条件などを考えると、かなりの技術力が要求されることを覚悟すべきでしょう。

ここでは、以上の点を踏まえて、PC9801シリーズの拡張スロットに挿入して使える、A-D/D-Aコンバータを含む音声処理に必要なすべての回路を、ワンボードで実現したものです。

分析周波数を4kHz、分解能を8ビットと抑えたために、A-D/D-Aコンバータに現在最もよく使われているタイプを選ぶことができ、コスト・パフォーマンスのよいものとすることができました。

分析周波数上限の4kHzという値は、音声の個人生情報に2kHz以上の長時間平均スペクトラムに含まれ

ているといわれていること<sup>(1)</sup>、およびほとんどの楽器の基本周波数がこの範囲に含まれること<sup>(2)</sup>などを考慮して、妥当な値だと考えています。

分解能に関しては、音声のダイナミック・レンジが50dB以上あるといわれていることから、10ビットとしたいところですが、データ量や計算処理のことを考えて、8ビットに決定しました。実際に使用した感じでは、マイクの使い方に気をつける点以外、特に問題にはならないようです。本装置のスペックを表2-1に示します。

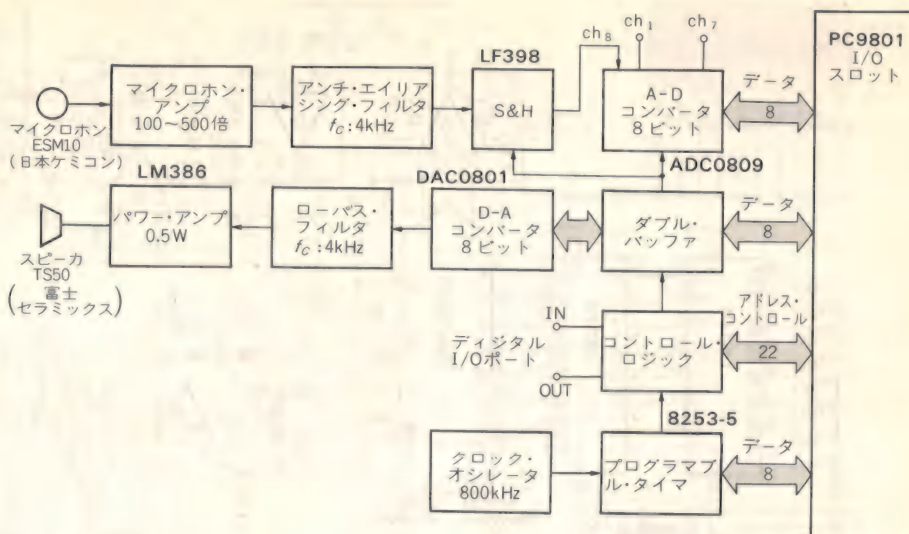
このボードは、コンピュータによる音声合成/音声認識のための開発ツールです。主な機能には、

- ▶ 音声データの取り込み
- ▶ 取り込んだデータの編集、各種数学演算
- ▶ ファイル・デバイスやスピーカへの出力

などがあります。これらの機能はソフトウェアで実現されており、ソース・リストで100ページ以上にわたって記述されているため、ここでは概要を述べるにとどめます。

図2-1に音声合成の開発フローチャートを示しますが、音声データの編集というのは、おおむねワード・プロセッサによるテキスト編集と同じ要領で行われます。つまり、波形データの任意の部分をグラフ表示し、置換、削除、挿入などの作業を行うものです。異なるのは、扱うデータ量が数十Kバイトから数百Kバイトと

〈図2-2〉  
音声ツール・ボードの  
ブロック・ダイヤグラム



いう、大量のデータである点です。

したがって、編集機能もそれに応じた処理速度、使いやすさが要求され、この点に関する配慮が欠けたものは全く使いものになりません。

このボード用ソフトウェアでは、ポインティング用の各種カーソル、多チャンネル表示、チャンネル間データ移動、X軸およびY軸のフレキシブルな設定、ワンキー・コマンドによる入力操作の簡易化などの機能の付加、強化により、ほとんど不便を感じないように作成してあります。

表2-2にそれらのコマンド一覧、図2-2に回路のブロック・ダイヤグラムを示します。

## 2-2 音声処理ボードの回路構成

それでは本ボードの回路構成について説明します。全体回路図を図2-3に示しますが、これをみながら各部の動作を理解してください。

### ● 入力部

入力にはエレクトレット・コンデンサ・マイクロホンが接続されます。このマイクロホンは、出力インピーダンスが非常に高く、FETによるインピーダンス変換回路が組み込まれ、その変換出力がラインに出力されます。

このため、FETを動作させるための電圧を信号ラインに加えるので、入力部は図2-4に示すような回路構成となります。

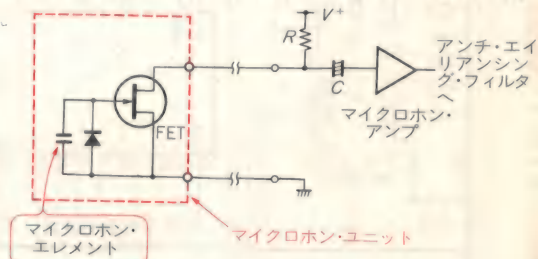
入力信号は、マイクロホン・アンプで100~500倍に増幅され、次のアンチ・エイリアシング・フィルタに出力されます。

### ● アンチ・エイリアシング・フィルタ

A-Dコンバータの応用で必ず問題となるのが、このフィルタの部分です。

市販の専用機では、精度とダイナミック・レンジの

〈図2-4〉 エレクトレット・コンデンサ・マイクロホンと入力部



確保のために、 $-100\text{dB/oct}$ 以上のロール・オフ(減衰)特性をもつフィルタが採用されているようです。ここでは、対象を音声、あるいは楽器音などに絞って、4次のパワース型フィルタで簡単にまとめてみました。

フィルタ特性は $-24\text{dB/oct}$ です。

### ● サンプル&ホールド回路

入力信号がA-D変換期間に変動すると、正しい変換値が得られないため、サンプル&ホールド回路で一定の入力電圧に保持します。

このボードのA-D変換スピードは $100\mu\text{s}$ /ワードで、この時間内でサンプル&ホールドとA-D変換を行わなければならない。

A-Dコンバータは、ADC0808/9を、クロック800kHzで使用し、変換時間を約 $90\mu\text{s}$ に縮めています。したがって、アキュイジション時間 $10\mu\text{s}$ で、8ビット分解能の $\frac{1}{2}\text{LSB}$ である0.2%の誤差に収まればよいということになり、ほとんどの市販専用ICが使えます。

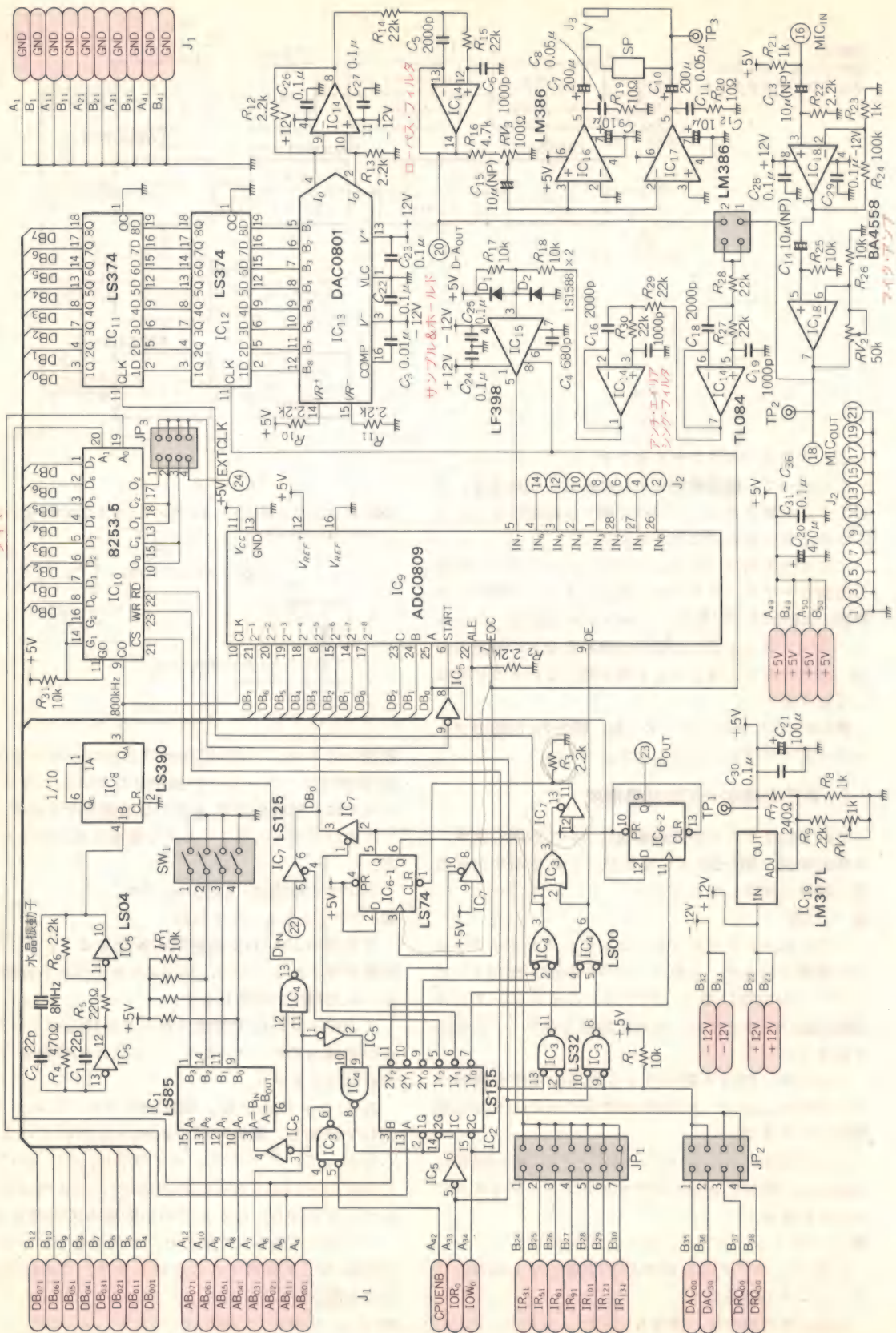
ここでは、最もよく使われ、コストも下がってきたLF398(NS)を採用することにします。このICの主な特性を図2-5に示します。

### ● A-Dコンバータ回路

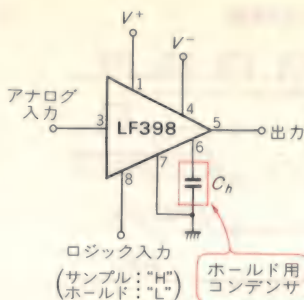


〈図2-3〉 音声処理ツール・ボードの全回路図

タイマ

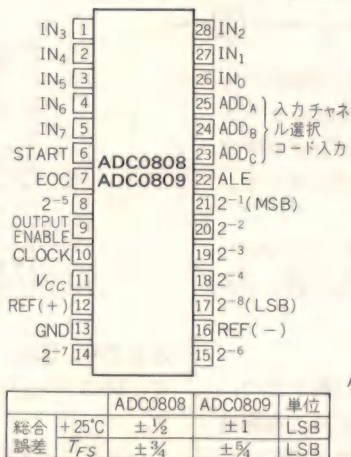


〈図2-5〉 サンプル&ホールドICの特性

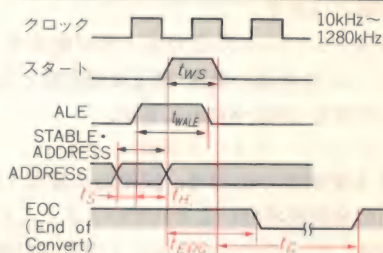


電源電圧	±18V(max)
入力オフセット電圧	2mV(typ)
入力バイアス電流	10nA(typ)
入力インピーダンス	10 <sup>10</sup> Ω(typ)
ゲイン・エラー	0.004%(typ)
リーク電流(ホールド時)	30pA(typ)
セトリング時間 (1mVに達するまでの時間)	0.8μs(typ)
アキュリゼーション時間 (0.1%減少するまでの時間, C <sub>h</sub> =1000pF)	4μs(typ)

〈図2-6〉  
8ビット8チャンネルA-Dコンバータ  
ADC0808/09の特性



$t_{WS}$	最小スタート・パルス幅	100ns( typ)
$t_{WALE}$	最小ALE/パルス幅	100ns( typ)
$t_S$	最小アドレス・セットアップ・タイム	25ns( typ)
$t_H$	最小アドレス・ホールド・タイム	25ns( typ)
$t_C$	変換時間	100μs( typ)
$t_{EOC}$	EOCデレイ・タイム	8clock+2μs



A-D変換速度が100μs程度の8ビットA-Dコンバータは種類も多く、コストも非常に安く手に入ります。このボードでは、CPUコンパチブルといわれるものの中から、C-MOS A-DコンバータADC0809(NS)を採用しました。主な特性を図2-6に示します。

このA-Dコンバータは、クロックが640kHzのとき、標準で100μs、最悪で116μsの変換時間となります。このボードのスペック100μs/ワードを保証するために、クロックを800kHzで使用することにします。

クロック800kHzでは、変換時間は約90μs(実測)となり、サンプル&ホールド回路のアキュリゼーション時間にかなり余裕が出てきます。

入力には8チャンネル分あり、チャンネル1から7までは直接入力、チャンネル8のみにフィルタおよびサンプル&ホールド回路を設けています。

直接入力の電圧範囲は0～5Vです。チャンネル8は、サンプル&ホールドの入力で、+2.5Vのオフセットを加えています。

チャンネルのセレクトは、デコーダとラッチ回路が内蔵されているので、3本のアドレス・ラインA、B、Cにラッチ・タイミング入力ALEとともに選択アドレスのデータを入力するだけです。

入力電圧のフルスケールの設定は、 $V_{ref}(+)$ および $V_{ref}(-)$ で行います。出力コード $N$ と入力電圧 $V_{in}$ の関係は(1)式のように表されます。

$$N = \frac{V_{in} - V_{ref}(+)}{V_{ref}(+) - V_{ref}(-)} \times 256 \dots\dots\dots(1)$$

このボードでは、 $V_{ref}(+)$ は安定化した+5V、 $V_{ref}(-)$ はGNDとしているために、4.98Vがフルスケールになります。 $V_{ref}(+)$ を5.02Vにセットすると、5.00Vがフルスケールになります。

A-D変換開始パルスは、プログラマブル・タイマ8253-5より得ており、100μsから1.25μsステップの任意時間のサンプリング時間が設定できます。

変換終了後、EOC(エンド・オブ・コンバージョン)出力が“L”から“H”へ変化します。この信号でサンプル&ホールド回路をホールド・モードからサンプル・モードへ切り替えます。同時にこの立ち上がりエッジで、データ・レディ・フリップフロップ(IC<sub>6-1</sub>)をセットします。

このフリップフロップの出力は、3ステート出力でデータ・バスに接続されているので、ソフトウェア・ポーリングにより、変換終了を知ることができます。ジャンパでこの出力がインタラプト入力に接続されている場合は、変換の終了時に割り込みをかけることができます。

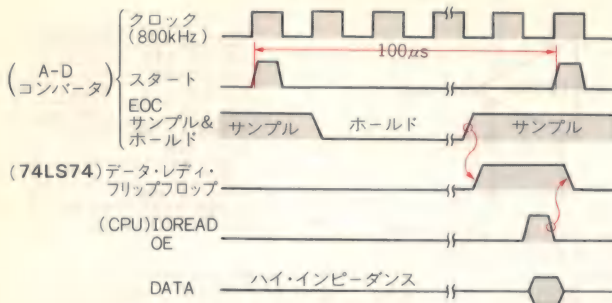
このようすを図2-7に示します。

### ● D-Aコンバータ回路

D-A変換は、通常ダブル・バッファを使用して、タイミングの確保およびグリッジの低減を図ります。



〈図2-7〉 A-Dコンバータの変換



このボードでは、DAC0801にラッチを2段設けて、D-Aコンバータを構成しています。1段目のラッチには、CPUのI/Oライト信号、2段目はプログラマブル・タイマのタイミング・パルスでラッチ・タイミングを得ています。

D-Aコンバータの出力モードが電流モードなので、OPアンプを差動の電流-電圧変換回路として使います。おもな特性を図2-8に示します。

#### ● ローパス・フィルタ回路

D-A変換後の出力は、段階的に変化しますので、多量の高調波成分が含まれています。これらの高調波成分は、すべてノイズとなって出力されますので、これをローパス・フィルタによって除去します。

フィルタ回路はアンチ・エイリアシング・フィルタに使用したものと同じで、カットオフ周波数4kHz、2次のパワース型フィルタです。

#### ● パワー・アンプ回路

ローパス・フィルタの出力は、レベル調整が行われた後、パワー・アンプへ入力され、スピーカをドライブします。

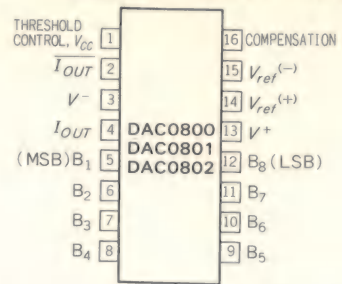
パワー・アンプには8ピン・ミニ・ディップ・タイプのLM386(NS)を採用しました。パワー・アンプの電源は、マイコンのロジック電源より得ているため、出力の振幅が十分に取れません。+5Vでは8Ω負荷で約0.2Wです。そのため、LM386を2個使用し、パレル出力回路として、約2倍の出力を得ています。

### 2-3 製作と調整

実装上の要点は、**アナログ回路とデジタル回路の分離**に気をつけるということに尽きます。特にマイクロホン・アンプ部は、ゲインが100倍以上あるので、グラウンド処理を誤ると、たちまち8ビット精度があまりくなります。調整は、

- ① 基準電源を+5Vに合わせる
- ② マイクロホン・アンプの出力が±2.5Vに収まるように、ゲインを設定する
- ③ スピーカの音量が適当になるように、出力アンプのポテンショメータを調節する

〈図2-8〉 8ビットD-AコンバータDAC0801の特性



電源電圧	±4.5V~±18V
セトリグ時間 (±1LSB以内に 収まる時間)	100ns (typ)
フルスケール温度係数	±10ppm/°C

\*DAC08の型名でも出ている

	DAC0800	0801	0802	単位
単調性 (max)	±0.19	±0.39	±0.1	% FS

以上です。③は、次に示すテスト・プログラムを組んで、D-Aコンバータに波形を入力することが必要です。

ハードウェアのチェックは次のようにしてください。

- ① IC<sub>8</sub>(LS390)の3ピンに、800kHzの方形波が出力されていることを確認。
- ② プログラマブル・タイマ8253を、テスト・プログラムの150行~170行のように、タイム・インターバル50msにセットする。IC<sub>9</sub>(ADC0809)の6ピンに、50ms間隔のパルスが入力されていることを確認。
- ③ マイクロホンからの入力、IC<sub>9</sub>の5ピンに、サンプル&ホールド後のステップ信号として入力されていることを確認。
- ④ I/OポートのD0h番地に00hを入力し、IC<sub>13</sub>の5ピン~12ピンがすべて“L”になることを確認。同様に、FFhのときすべて“H”になることを確認。

### 2-4 テスト・プログラム

このボードの機能を100%生かすためには、コンパイラやアセンブラを使用することになりますが、ちょっとしたテスト・プログラムなどには、BASICインタプリタで十分です。テスト・プログラムをリスト2-1に示します。内容は、

- ① デジタル入出力ポートの操作
- ② 三角波を発生させ、D-Aコンバータの出力をする
- ③ D-Aコンバータの出力の三角波をA-D変換し、数値で表示する

というものです。あらかじめデジタル・ポートの入力と出力、およびD-Aコンバータ出力とアンチ・エイ

## 〈リスト2-1〉テスト・プログラム

```

60 '
70 '
80 DEFINIT A-J
90 '
100 ' *** TIMER TEST ROUTINE *** ← 1秒間隔にビープ音を10回発生させる
110 '
120 PRINT
130 PRINT "PROGRAMMABLE TIMER TEST"
140 *TIMER
150 OUT &HDE, &H34      '** MODE 2, 16BIT BINARY COUNTER SELECT
160 OUT &HDB, &H40      '** SET 50msec. (80K/4000)
170 OUT &HDB, &H9C      '** 4000 = &h9C40
180 FOR I=1 TO 10
190   GOSUB *T50
200   BEEP
210 NEXT I
220 PRINT "TEST DONE"
230 GOTO *CHANNEL7
240 '
250 '
260 *T50
270 FOR J=1 TO 20
280   A=INP(&HD2)
290   IF (A AND 1) = 0 GOTO 280
300 NEXT J
310 RETURN
320 '
330 '
340 ' *** SELECT CHANNEL 7 ***
350 '
360 *CHANNEL7
370 OUT &HD2, 7
380 '
390 '
400 ' *** 1BIT DIGITAL I/O PORT TEST ROUTINE *** ← デジタル・ポートの出力
410 '                                     ビットを変化させ、ディ
420 *DIO                                     ジタル入力ポートで読み出
430 PRINT                                     し、一致していることを確
440 PRINT "DIGITAL I/O PORT TEST"           認、不一致のときは、エラ
450 FOR I=0 TO 100                           ー・メッセージを表示、
460   OUT &HD4, 0                             100回繰り返す
470   A=INP(&HD4)
480   IF (A AND 1) <> 0 GOTO *DIOERROR
490   OUT &HD4, 1
500   A=INP(&HD4)
510   IF (A AND 1) <> 1 GOTO *DIOERROR
520 NEXT I
530 PRINT "TEST DONE"
540 '
550 '
560 ' *** A-D, D-A CONVERTER TEST ROUTINE ***
570 '
580 *ADCON ← D-Aコンバータの入力を0から255ま
590 PRINT                                     でインクリメントさせ、そのアナログ
600 PRINT "A-D, D-A CONVERTER TEST"         出力をA-Dコンバータへ入力し、変換
610 OUT &HD0, 0                             結果を数値で表示する。
620 A=INP(&HD2)                             '** CHECK TIMER FLAG
630 IF (A AND 1) = 0 GOTO 620
640 A=INP(&HD0)
650 FOR I=0 TO 255
660   OUT &HD0, I
670   A=INP(&HD2)                             '** CHECK TIMER FLAG
680   IF (A AND 1) = 0 GOTO 670
690   A=INP(&HD0)
700   PRINT HEX$(A);SPC(2);
710 NEXT I
720 OUT &HD0, 128                             '** SET DAOUT=0
730 PRINT
740 PRINT "TEST END"
750 '
760 END
770 '
780 '
790 *DIOERROR ← デジタルI/Oポート
800 PRINT "DIGITAL I/O PORT ERROR"           (エラー・メッセージ)
810 END
820 '

```

リアシング・フィルタの入力を接続してから、プログラムをスタートさせます。ポート・アサインを表2-3に示します。

〈表2-3〉I/Oポートのアサイン

アドレス	リード	ライト
× 0H	A-Dコンバータのリード	D-Aコンバータのライト
× 2H	タイマ・フラグ (b7)	A-Dチャネル・セレクト (b0~b2)
× 4H	デジタル入力 (b0)	デジタル出力 (b0)
× 8~× EH	プログラマブル・タイマ	プログラマブル・タイマ

(注) ×は、ディップ・スイッチで設定される上位4ビット・アドレス

## ●参考・引用・文献

- (1) サイエンス、数理科学、1984年6月号No252, p. 33.
- (2) 丸善、理科年表、東京天文台編、物85.
- (3) 佐藤清忠：パソコンとA-Dコンバータのインターフェース技術、トランジスタ技術、1984年2月号, p. 309.
- (4) National Semiconductor Data Conversion/Acquisition Data Book, 1984年.



### ③ロジック・アナライザの製作

電子回路を設計するときのトラブルの原因として、タイミングずれなどの現象があります。このタイミングずれが繰り返し現象である場合は、オシロスコープで観測することができますが、単発現象の場合はそうはいきません。

単発現象を観測するためには、ロジック・アナライザがありますが、高価で一般ビギナにはとても使えるものではありません。それに、コンピュータに接続し、簡易言語で自分本位の解析方式にレベルアップできるシステムにするには、何百万円もかかります。

そこで、一般に手に入るパソコンと接続し、データ解析システムまで発展できるロジック・アナライザを考えてみました。

#### 3-1 ロジック・アナライザの原理

最初に考えたのは48KバイトのS-RAMボードを用い、カウンタでアドレス・アップしていく実に簡単な構成のものです。

48Kバイトすべてをパソコンのメモリ上に転送するのも面倒なので、ソフトウェア上で効率よくデータをつめていきました。

配列を2種類、つまりサンプルしたデータ値の配列と、時間データ値の配列を用意し、データとしてもっとも古いデータから、次々にデータ比較を行い、データが同じならばそのデータは捨てて、データが違っていたら(データの変化点ならば)もっとも古いデータからのクロック数を時間データの配列に、またその時のサンプルしたデータ値を、サンプル・データの配列に加えていくと、データは実に効率よくメモリ内に納まっていきます。

こうすることによって、パソコンでのデータ検索を楽に行うことができます。ただし、当時はRAMに6116を用いたために、せいぜい200nsでサンプリングするのが限度でした。

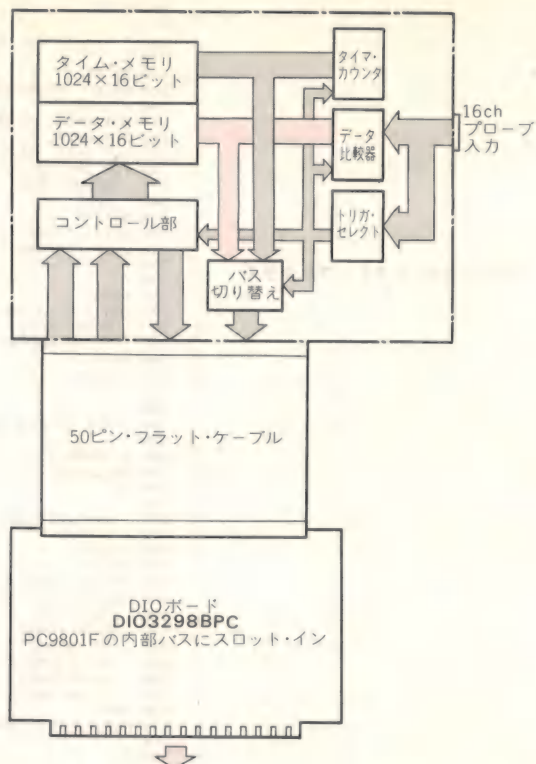
次に製作したのは、以前のようなソフトウェアで行ったことをハードウェア上に置き換え、なおかつRAMに2114の高速バージョンであるHM6148HP-35を用い、50ns20MHzのサンプリングを目標に製作しました。

#### 3-2 ロジック・アナライザの構成

パソコンは、PC9801Fを用いました。I/Oボードには市販品(サイエンス社:DIO3298B)を用いましたが、回路構成は簡単ですので自作も可能です。回路図は後で示します。

データの取り込み方式には前記の**変化点サンプリング**

〈図3-1〉システム構成図



と、従来の**基本クロックごとに取り込むフル・サンプリング**の2種類があります。

GPIBのバス・モニタやA-Dコンバータなどへの拡張性を考えて、**入力**は**16ch/TTLレベル**とし、測定条件などはすべてパソコン側からソフトウェアで指定できるようにしました。

システム構成を図3-1に、システム仕様を表3-1に示します。基板は、コントロール部とメモリ部の2枚に分けました。

#### 3-3 コントロール基板の回路構成

コントロール・ボードのブロック図を図3-2に、回路図を図3-3に示します。クロック・オシレータには、出力の安定な20MHzのクロック・モジュールを使用しました。

また、サンプリング・クロックは、図3-3に示すようにクロック・オシレータの出力を分周し、 $\times 1$ 、 $\times 1/2$ 、 $\times 1/4$ の3種類と、 $\times 1$ 、 $\times 1/10$ 、 $\times 1/100$ 、 $\times 1/1000$ 、 $\times 1/10000$ の5種類を組み合わせると15種類より選定することができます。また、**外部クロック入力も可能**にしています。外部クロックは、パルス幅の“L”レベルが25ns以上は必要ですので注意してください。

制御部のカウンタは、プリトリガを行うためのカウンタで、原則的にデータはトリガ以前のデータを128

〈表3-1〉 システムの仕様

- (1) データ入力: 16ch.....スレッショルド TTLレベル(+1.4V±0.2V)  
 (2) トリガ入力: 外部入力.....最小パルス幅 100ns(min)  
 内部入力.....1ch~16ch中、任意の1ch  
 最小パルス幅 100ns(min)
- \*トリガの内部、外部入力切り替えおよび内部トリガの入力chの選択は、キーボードから任意指定可能。
- (3) トリガ極性: 両極性.....トリガ入力の内部、外部にかかわらず、立ち上がり(+)/立ち下がり(-)をキーボードからの任意指定可能。
- (4) サンプル・クロック: 外部入力.....50ns以上(20MHz以下)  
 内部入力.....15種類
- |              |              |             |
|--------------|--------------|-------------|
| 50 ns,       | 100 ns,      | 200 ns      |
| 500 ns,      | 1 $\mu$ s,   | 2 $\mu$ s   |
| 5 $\mu$ s,   | 10 $\mu$ s,  | 20 $\mu$ s  |
| 50 $\mu$ s,  | 100 $\mu$ s, | 200 $\mu$ s |
| 500 $\mu$ s, | 1 ms,        | 2 ms        |
- \*サンプル・クロックの内部、外部入力切り替えおよび内部クロック周期の切り替えは、キーボードから任意指定可能。
- (5) 最小サンプリング可能パルス幅: サンプリング間隔+10ns  
 (6) メモリ・サイズ: データ・メモリ.....1024×16ビット  
 タイム・メモリ.....1024×16ビット
- (7) サンプリング方式:
- ① 変化点サンプリング  
 データの変化点のみ、その時の時間とサンプル・データをメモリに記憶する。1 Kバイトのメモリ容量で、最大64 Kバイトに相当する時間区間をサンプルできる。
  - ② フル・サンプリング  
 従来のロジック・アナライザのサンプリング方式、サンプル・クロックごとにメモリに記憶する。
- \*キーボードからの任意指定可能。
- (8) データ・ポジション:
- ① 変化点サンプリング時  
 トリガ前 128 変化点、以後データ変化点の密度によって取り込み数は変化する (64 K×サンプル・クロック以内)
  - ② フル・サンプリング時  
 トリガ以前 128 サンプル、以後 896 サンプル

バイト、トリガ以後896バイトを確保して、測定を終了します。

本システムには2種類のサンプリング方式があります。

#### ① フル・サンプリング方式

この方式は1クロックごとにメモリに取り込む、ロジック・アナライザ本来の方式です。

#### ② 変化点サンプリング方式

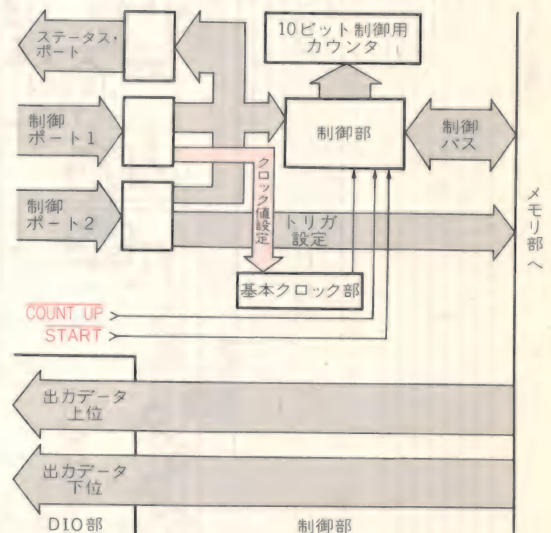
1クロックごとに前回のデータと比較し、同じならばメモリに書き込みます。違っていた時に限って、その時の時間データとともに、サンプリング・データを書き込みます。メモリ基板には16ビットのタイム・カウンタをもっているため、変化点が896バイト以下の場合、実に64Kバイトのメモリに相当する時間幅を、本システムは取り込んだことになります。

### 3-4 メモリ基板の回路構成

図3-4にメモリ基板のブロック図を図3-5には回路図を示します。

トリガ入力は、内部トリガでは図のように74150を用い、16chの中の1ビットを任意に指定できます。また外部トリガ指定もできます。トリガは、コントロール・ボード上で、74F86によって任意に論理反転できるようにになっています。

〈図3-2〉 コントロール部のブロック図



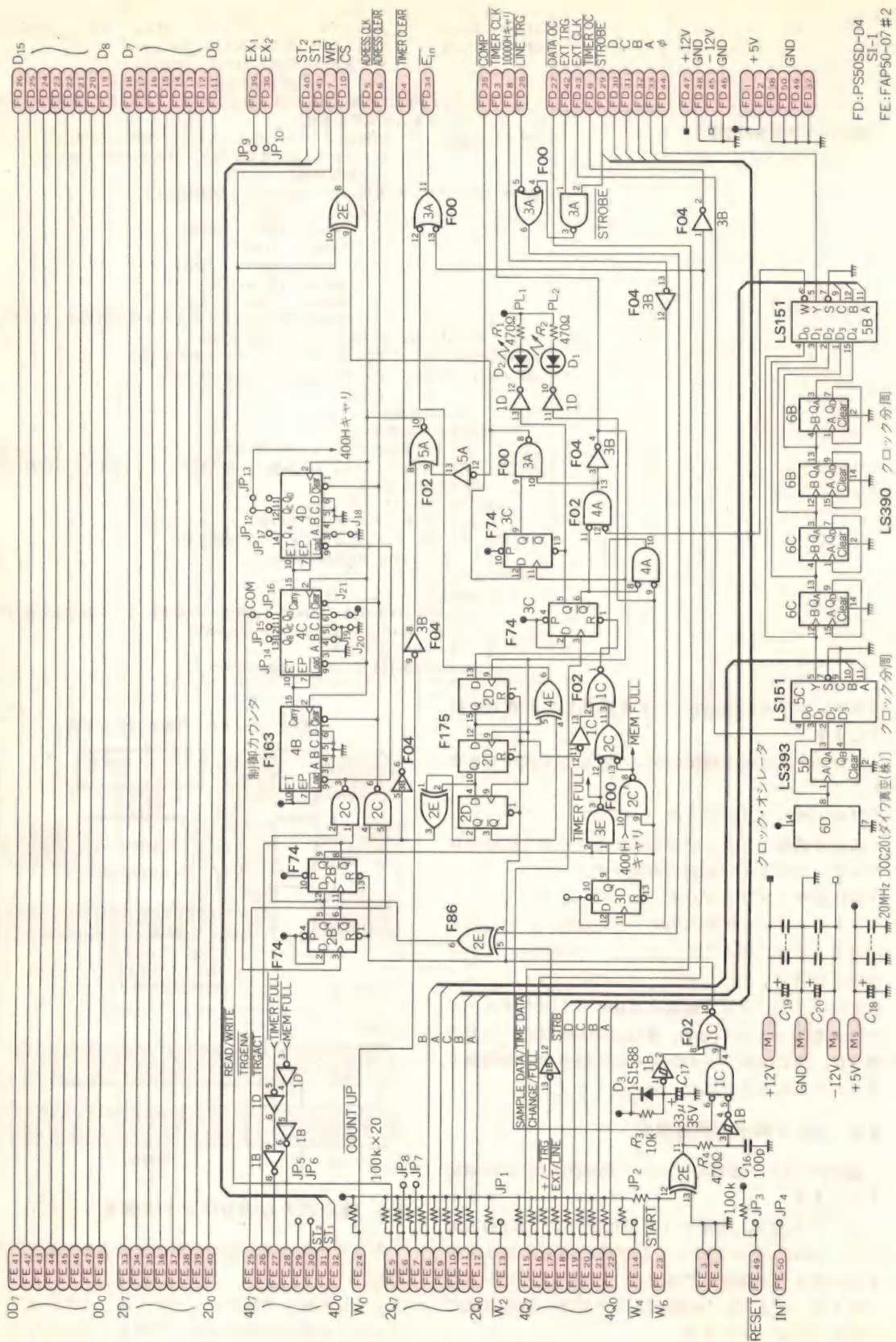
### 3-5 本システムのカウンタの働き

本システムには、三つのカウンタがあります。すなわち、

- ① メモリ基板上の16ビット・タイム・カウンタ
- ② メモリ基板上の10ビット・アドレス・カウンタ

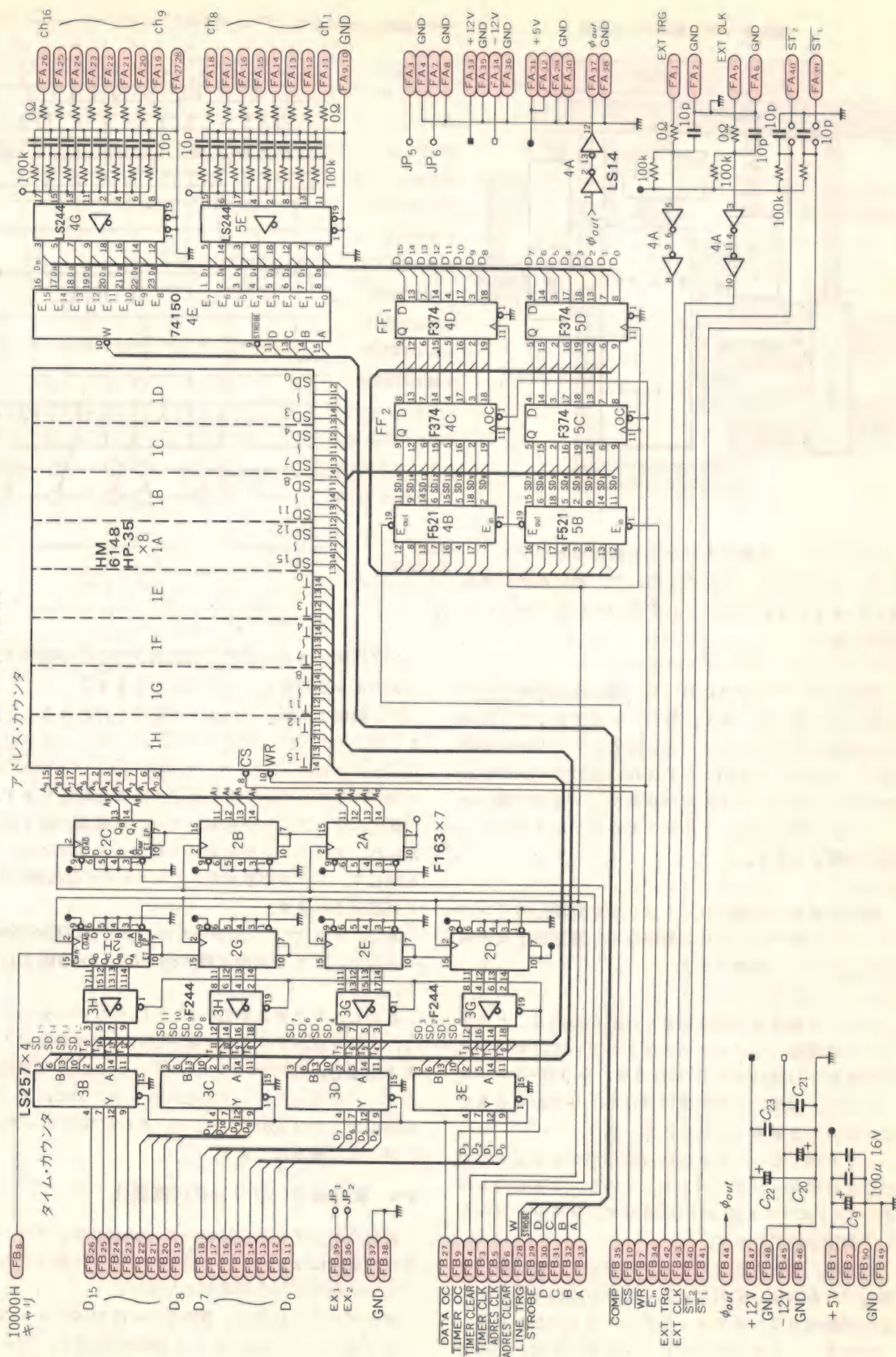


図3-3の回路図



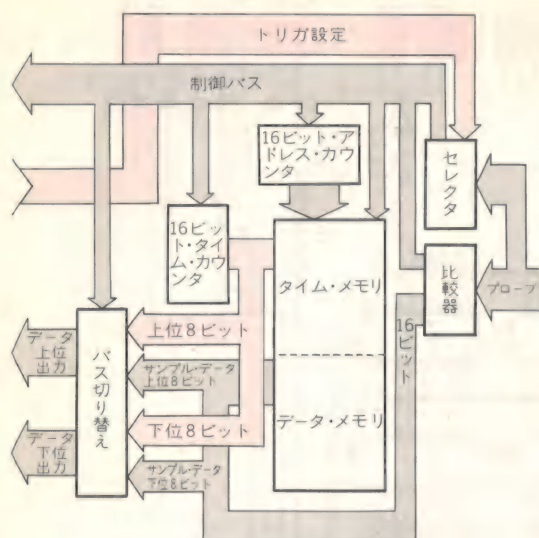
FD:PS50SD-D4  
SI-1  
FE:FAP50-07 #1

＜図3-5＞ メモリ・ボードの回路図





〈図3-4〉 メモリ部のブロック図



③コントロール基板上の10ビット制御用カウンタです。この三つのカウンタを用いて、変化点サンプルを行います。それでは、それぞれのカウンタについて説明します。

#### ①16ビット・タイム・カウンタ

変化点サンプリングにおいて、変化点の時間データを発生させるカウンタで、毎クロックごとの立ち上がり同期してインクリメントします。トリガがきた時点で、0にカウンタがロードされ、次のクロックから次々にインクリメントしていきます。トリガ以後、カウンタが一周したら、タイマ・カウント・フルとして測定を終了します。

#### ②10ビット・アドレス・カウンタ

変化点があった時だけ、クロックに同期してインクリメントします。キャリに関係なく、測定終了まで何周でもカウントを続けます。

#### ③10ビット制御用カウンタ

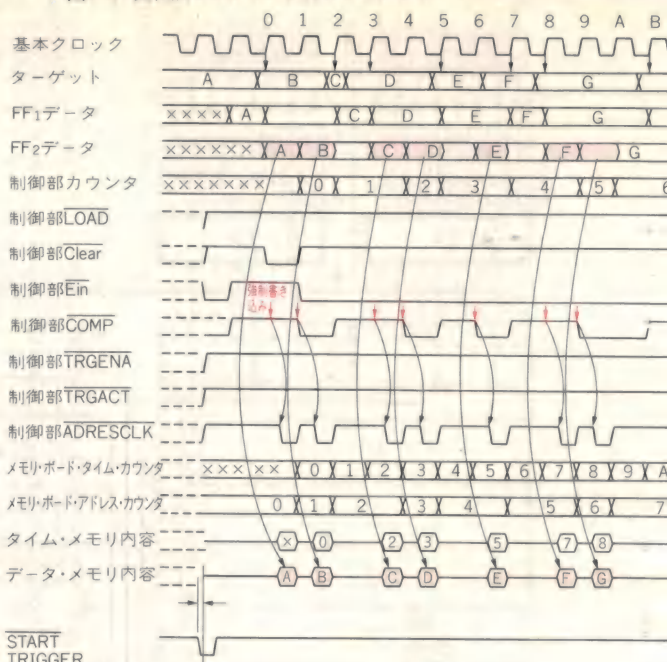
スタート時は0hから始まり、変化があるごとにクロックに同期してインクリメントしていき、カウンタが080h(128バイト分)になると、トリガ・イネーブルにして、以後トリガを受け付けてトリガがくるまで080hのままホールドされています。

トリガがくると、それ以後は変化点があるごとに080hに次々インクリメントしていき、0400hのキャリ(1Kバイト分)が出た時点で、メモリ・フルとして測定を終了します。

つまり、変化点サンプリングにおいて、測定終了を決定づけるのは、①のタイム・カウンタのキャリか、③の制御用カウンタのキャリということになります。

ちなみに、フル・サンプリングにおいては、①のカ

—— 〈図3-7〉 変化点サンプリング時のタイムチャート



スタート地点

ウンタは用いませので、測定終了は③の制御用カウンタのキャリだけということになります。

少し面倒ですが、このような動作を行わせることによって、フル・サンプリングにおいては、トリガ以前のデータが128バイト、トリガ以後のデータが896バイト確実にそろって、測定を終了することになります。

変化点サンプリングにおいては、2種類の終了が考えられ、メモリ・フルにおいてはフル・サンプリングと同じで、トリガ以前128バイト、トリガ以後896バイトが確保されます。

また、タイマ・フルにおいては、トリガ以前が128バイト以上、トリガ以後は最大896バイトが確保されます。

しかし、トリガ以前128バイト以上のデータについて、トリガ点がトリガ・イネーブルとほぼ同時に入ってきた場合、128バイト以前のデータには前回測定したデータが残っている可能性もありますので、トリガ以前のデータは128バイトとし、それ以前のデータは、ソフト上で読み捨てます。

### 3-6 変化点サンプリングの高速化

変化点サンプリングにおいて、50nsのサンプリングを行う場合には、変化点の検出からメモリ書き込みまでを50ns以内に行う必要があります。

またデータの比較は、図3-5における16ビットのフリップフロップ(74F374)を2個直列接続し、同一ク





〈表3-2〉 ブローブ側コネクタ表(40ピン)

I/O	信号名称	ピン番号		信号名称	I/O
IN	EXT TRG	1	2	GND	OUT
	NC	3	4	GND	
IN	EXT CLK	5	6	GND	
	NC	7	8	GND	
OUT	GND	9	10	GND	IN
	ch <sub>1</sub>	11	12	ch <sub>2</sub>	
	ch <sub>3</sub>	13	14	ch <sub>4</sub>	
	ch <sub>5</sub>	15	16	ch <sub>6</sub>	
	ch <sub>7</sub>	17	18	ch <sub>8</sub>	
	ch <sub>9</sub>	19	20	ch <sub>10</sub>	
	ch <sub>11</sub>	21	22	ch <sub>12</sub>	
	ch <sub>13</sub>	23	24	ch <sub>14</sub>	
	ch <sub>15</sub>	25	26	ch <sub>16</sub>	
OUT	GND	27	28	GND	OUT
	GND	29	30	GND	
	+5V出力	31	32	+5V出力	
	+12V出力	33	34	-12V出力	
	GND	35	36	GND	
	CLK OUT	37	38	GND	
IN	STI <sub>1</sub>	39	40	STI <sub>2</sub>	IN

\*使用コネクタ：山一製 FAP-40-07. #2

\*適合プラグ：山一製 FAS-40-17(フラット・ケーブル用)  
：山一製 UFS-40B-04およびUFSコンタクト  
(ユニフレックス用)

〈表3-5〉 各端子の機能(ブローブ側)

ch <sub>1</sub> ~ ch <sub>16</sub>	本器のブローブ入力端子
STI <sub>1</sub> , STI <sub>2</sub>	ブローブからのステータス入力端子
CLK OUT	本器のサンプリング・クロック出力
EXT TRG	外部トリガの入力端子
EXT CLK	外部クロックの入力端子

\*スレッショルドは、すべてTTLレベル

います。

データ設定→測定→データ取り込み→データ処理までのフローチャートを図3-9に示します。図3-10に、それぞれの取り込み方式におけるデータの状態を示します。

図(a)はフル・サンプリングにおいて測定終了した場合で、メモリは測定終了した時点が最も新しいデータで、アドレスを1024番地とします。アドレス・カウンタを一つアップすると、最も古いデータが現れます。ここをアドレス1番地とします。順々にアドレスをインクリメントしていきまると、アドレス128番地目がトリガ点です。

図(b)は、変化点サンプリングにおけるメモリ・フル

〈表3-3〉 DIO側コネクタ表(50ピン)

ADRS=1AD0H				IO IN/OUT	I/O アドレス
ピン 番号	ロジック・アナライザ信号名称				
1, 2	NC			+ 5 V	+ 5 V
3, 4	GND			GND	GND
5	READ/WRITE			MSB	制御用ポート I (OUTPUT)  LSB  ADRS +2
6	NC				
7	NC				
8	B	クロック分周 (B)			
9	A				
10	C	クロック分周 (A)			
11	B				
12	A				
13	NC				
14	NC				
15	SAMPLE DATA/TIME DATA			MSB	制御用ポート II (OUTPUT)  LSB  ADRS +4
16	CHANGE/FULL				
17	+TRG/-TRG				
18	EXT/LINE				
19	D	内部トリガ入力			
20	C				
21	B				
22	A				
23	START			(OUTPUT)	ADRS+6
24	COUNT UP			(OUTPUT)	ADRS+0
25	TRG ENABLE	内部ステータス 入力	MSB	ステータス・ ポート (INPUT)  LSB  ADRS +4	
26	TRG ACT				
27	TIMER FULL				
28	MEM FULL				
29	NC				
30	NC				
31	ST <sub>2</sub>	ブローブ ステータス入力			
32	ST <sub>1</sub>		LSB		
33	DO <sub>7</sub>			MSB	"L"データ・ ポート (INPUT)  ADRS +2
34	DO <sub>6</sub>				
35	DO <sub>5</sub>				
36	DO <sub>4</sub>				
37	DO <sub>3</sub>				
38	DO <sub>2</sub>				
39	DO <sub>1</sub>				
40	DO <sub>0</sub>			LSB	
41	DO <sub>15</sub>			MSB	"H"データ・ ポート (INPUT)  ADRS +0
42	DO <sub>14</sub>				
43	DO <sub>13</sub>				
44	DO <sub>12</sub>				
45	DO <sub>11</sub>				
46	DO <sub>10</sub>				
47	DO <sub>9</sub>				
48	DO <sub>8</sub>			LSB	
49	NC				
50	NC				

\*使用コネクタ：山一製 FAP-50-07. #2

〈表3-4〉各端子の機能 (DIO側)

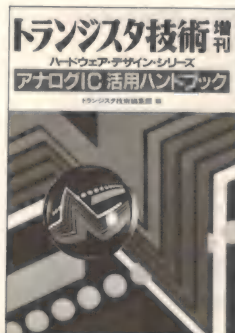
READ/WRITE	データ読み出しモードおよび測定モードを指定する。
クロック分周 (A), (B)	サンプル・クロックの内部、外部および分周比を設定する。
SAMPLE DATA/ TIME DATA	データ出力バスの内容を切り替えるための出力制御を行う。
CHANGE/FULL	変化点サンプリングとフル・サンプリングの切り替えを行う
+TRG/-TRG	トリガの内部、外部にかかわらず、トリガ極性(立ち上がり/ 立ち下がり)の切り替えを行う。
EXT/LINE	トリガ入力を内部にするか外部にするかを切り替える
内部トリガ・セレクト	内部トリガのチャンネルを設定する。
START	200ns (min) の負のパルスを印加することにより、測定モード 時には測定を開始する。
COUNT UP	読み出しモードの時に本器内部のアドレス・カウンタをイン クリメントする。
TRG ENABLE	本器からの測定ステータス出力で、メモリに128バイト書き込 むと“L”になり、これ以後トリガの受け付けを開始する。
TRG ACT	本器からの測定ステータス出力で、TRG ENABLE後に、トリ ガが来ると“L”になる。
TIMER FULL	本器からの測定ステータス出力で、変化点サンプリング時の 測定で、変化点数が合計 1024 バイトにみたく、タイム・カウ ンタのOVERによって、測定を終了した場合に“L”になる。
MEM FULL	本器からの測定ステータス出力で、フル・サンプリング時の 測定終了時、または変化点サンプリング時の測定で、変化点 数が合計 1024 バイトになって、測定を終了した場合に“L” になる。
ST <sub>1</sub> , ST <sub>2</sub>	ブロープからのステータス出力。
DO <sub>0</sub> ~DO <sub>15</sub>	測定したサンプリング・データおよびタイム・データを出力。

(a) DIO側コネクタ (50ピン)

で終了した場合です。図(a)と同様、アドレス1番地からインクリメントしていくと、ちょうど129バイト後にタイム・データの内容が“0”である点が存在します。この0データの1バイト前がトリガ点になります。

図(c)は、変化点サンプリングにおけるタイマ・カウンタ・フルで終了した場合です。STOPした次をアドレス1として、タイム・データの内容が“0”である点で、なおかつ最もアドレスの高い点の1バイト前がトリガ点です。それ以前の128バイトを、最も古いデ

絶賛発売中!



CQ出版社

## ハードウェア・デザイン・シリーズ

★電子回路部品活用ハンドブックにつづく第2弾!

# アナログIC 活用ハンドブック

トランジスタ技術 増刊

2色刷

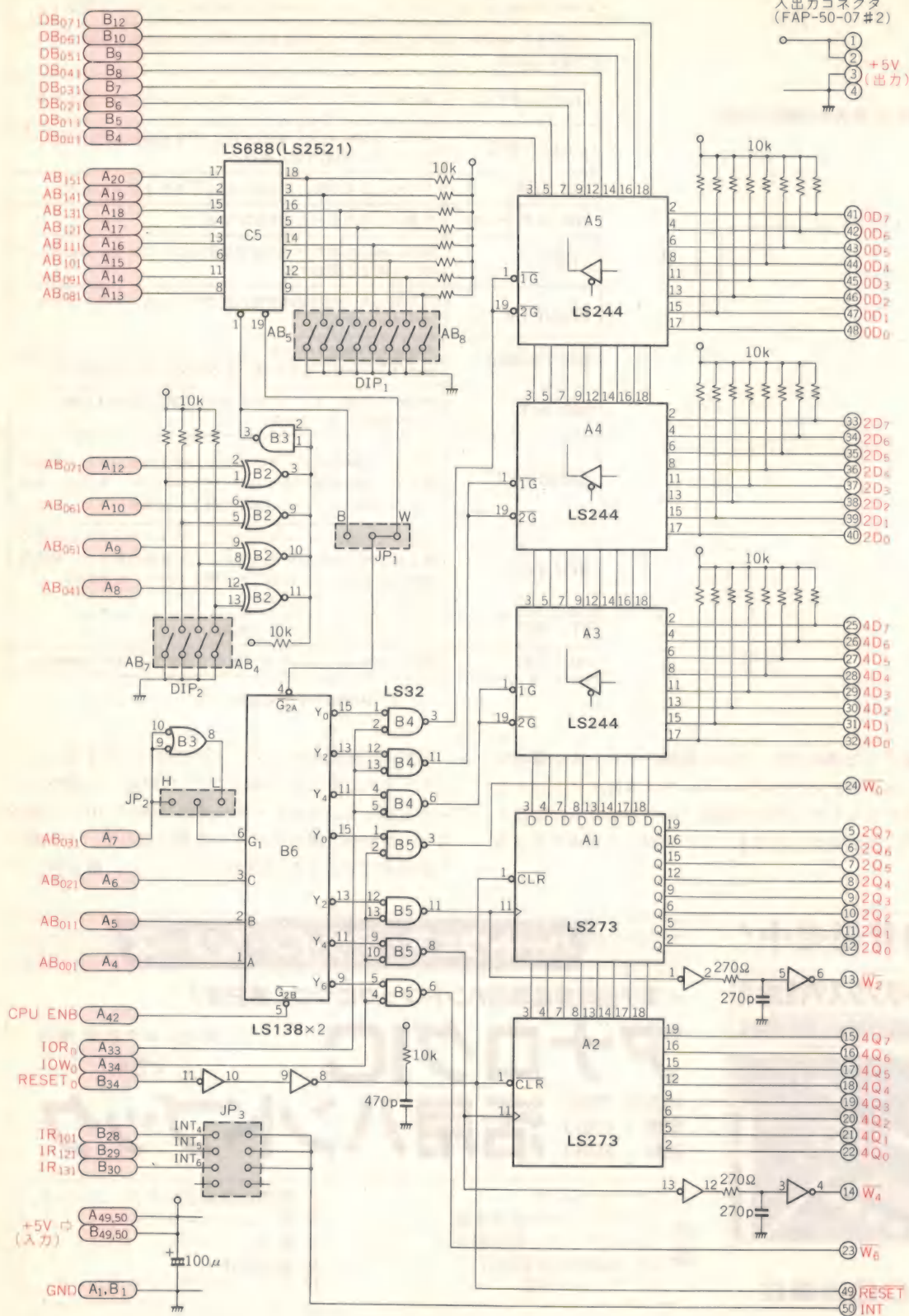
B5判 320頁  
定価 1,800円  
送料 300円

内容

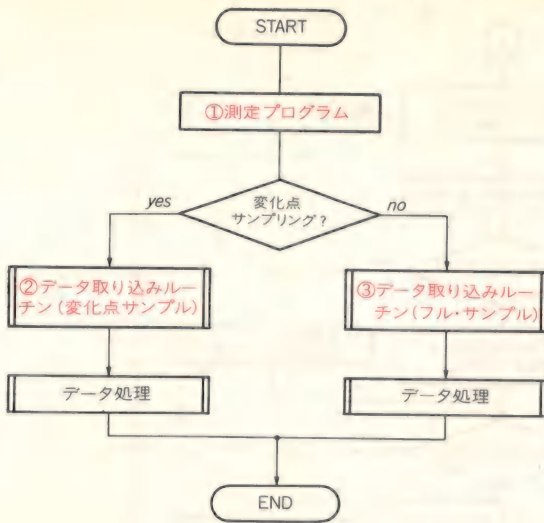
- (1) プロローグ
- (2) OPアンプの基本機能
- (3) OPアンプと線形回路
- (4) 非線形の演算回路
- (5) フィルタ回路
- (6) 発振回路およびV-Fコンバータ
- (7) D-Aコンバータ
- (8) A-Dコンバータ
- (9) 電源用IC
- (10) スイッチング・レギュレータ



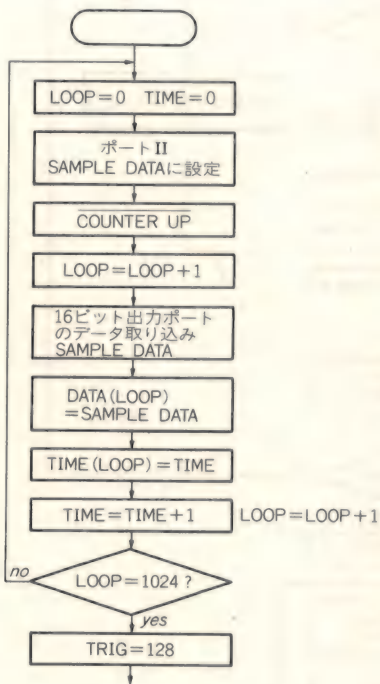
入出力コネクタ  
(FAP-50-07 #2)



〈図3-9〉 データ処理のフローチャート



(a) メイン・ルーチン



(d) ③データ取り込みルーチン  
(フル・サンプル)

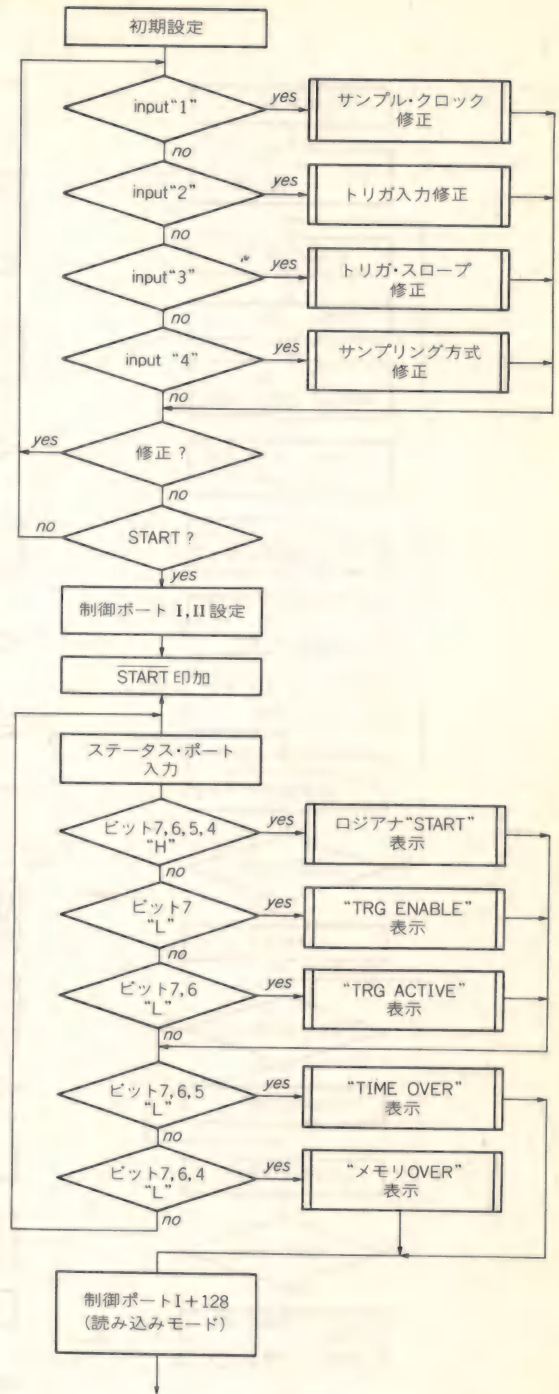
ータとみなします。

この場合のトリガ点以後のデータは896バイト以下で、ちなみに50nsのフル・サンプルング・モードにおいて、取り込み時の時間幅は、

$$50\text{ns} \times 1024 \approx 51.2\mu\text{s}$$

であるのに対し、この場合の取り込み時間幅は、

$$50\text{ns} \times 65536 \approx 3.2768\text{ms}$$



(b) ①測定プログラムのフローチャート

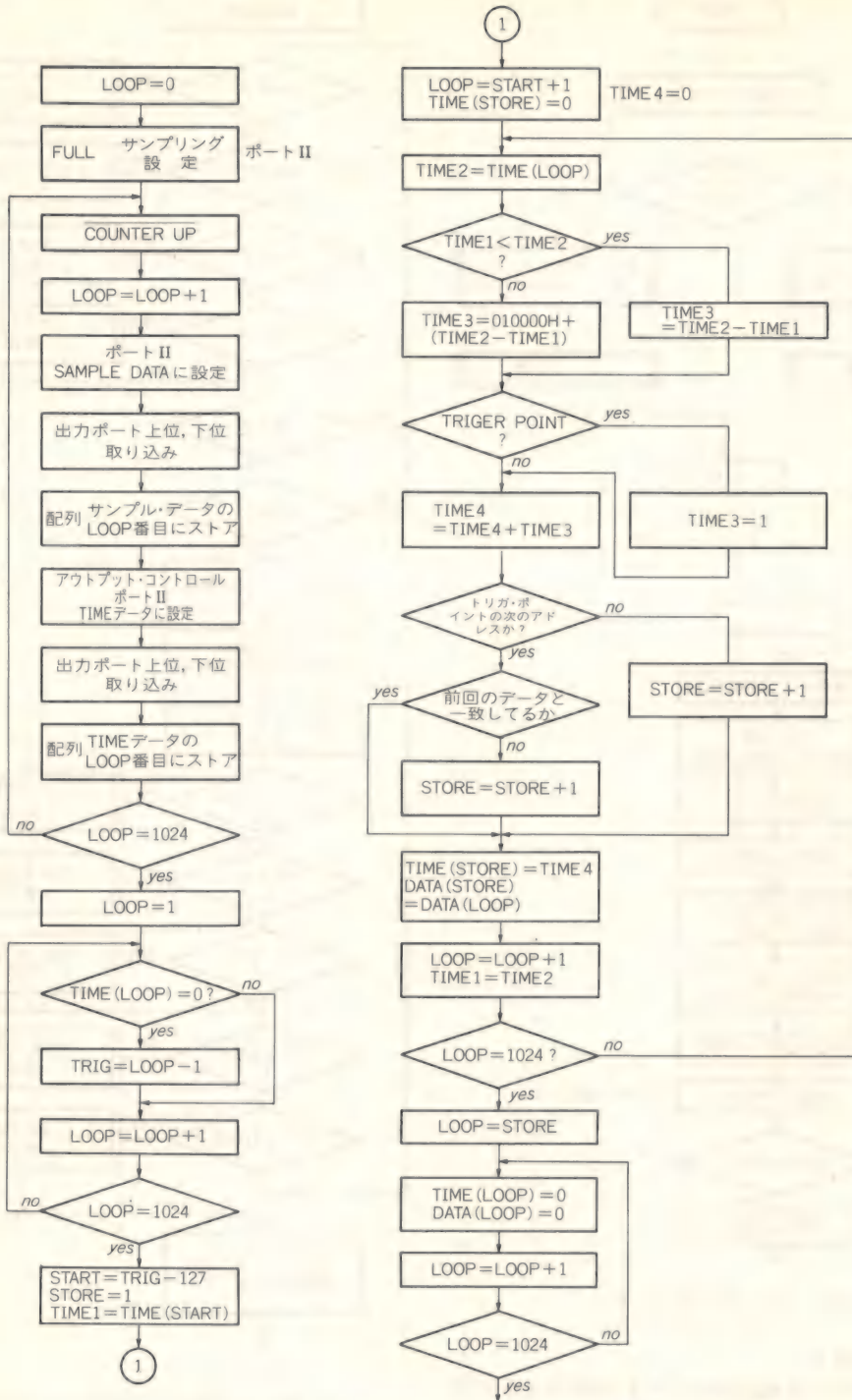
の時間幅(64Kバイトのメモリ)に相当します。

### 3-9 ノイズ対策

50nsサンプルングは筆者には初めてのことで、

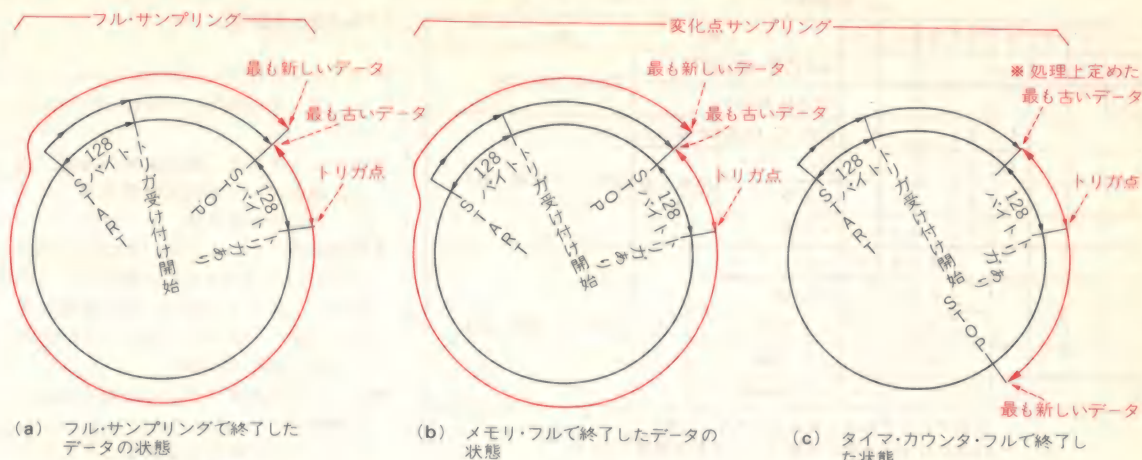


〈図3-9〉 データ処理のフローチャート(つづき)



(c) ②データ取り込みルーチン(変化点サンプル)

〈図3-10〉 データの状態



DATA VALID

※：トリガ受け付け開始から実際のトリガが来るまでの遅れが0の場合も考えられるので、データの確実性からトリガ点より128バイト前と定める。

ノイズに対してはかなり悩まされました。さらに今回は、メモリ・ボードと制御ボードを2枚に分けて製作したため、アース・ラインの電位差も影響しました。

ここでは実際に16chのバイナリ・カウンタを作って実験してみました。測定結果として、**プローブの長さを50cmぐらい引き回すと、チャンネルごとの遅れが確認されました。**

それから外部トリガにおける誤動作もあり、特にデータがFF FF hから0 hに変化する時など、プローブのアース・ラインにノイズが乗りました。アースの強化という意味では、信号線16本の場合、それと対になるアース16本を設けるのが理想なのですが、それだとツールとして使いづらくなってしまいます。

この対策としては、入力インピーダンスを高くするとかプローブを短くする、トリガ・ラインのグラウンドと信号のグラウンドを分ける、最小サンプリング・パルス幅がつかぬ程度に、信号ラインにコンデンサを入れる、などが考えられます。

そこでここでは、プローブの長さを30cmに定め、10 pF程度の容量をもたせ、プローブのアース・ラインの強化を行いました。その結果、誤動作を防ぐことができました。

### 3-10 制御プログラム

プログラムは、BASICで書いてあります。PC9801は、BASICから制御できるI/O空間が少なく、I/Oをこの少ないアドレスに割り当てるのは実にもったいないことですので、アドレスは1AD0h~1ADFhに置くことにしました。これによってBASICレベルにおけるI/O空間をつぶさずにすむことになります。ただしこのアドレスは、マシン語レベルで呼ばれることになります。

測定したデータは、決められた配列に入られますので、そのデータを利用して自分なりの方法でデータ処理を行えば、もっといろいろな解析が可能になっていきます。

最後になりましたが、各ポートの機能を表3-6に示しましたので、参考にして下さい。

#### 参考文献・引用文献

- (1) 岡村勉夫：標準デジタル・バス (IEEE-488) とその応用，CQ出版社。
- (2) サイエンス欄，DIO-3298BPC，DIO-3020PC，取り扱い説明書および回路図。
- (3) 日本電気欄，PC9801Fユーザーズ・マニュアル。
- (4) 鶴野和孝：GPIB信号チェッカの製作，トランジスタ技術，1984年4月号，p. 457。



▶▶▶ 新シリーズ CORE BOOKS 登場!! ▶▶▶▶▶

## 実用インターフェース設計法

—— マイコン活用のためのハードウェア技術入門 ——

畔津明仁 著，A5判，212頁，定価 1,400円



〈表3-6〉 各ポートの機能

① 制御ポート I

7	6	5	4	3	2	1	0	機 能	
0								WRITE MODE	READ/WRITE
1								READ MODE	
			0	0				EXT CLOCK	外部入力
			0	1				× 1	クロック分周 (B)
			1	0				× 1/2	
			1	1				× 1/4	
					0	0	0	× 1	(A) クロック分周 (A)
					0	0	1	× 1/10	
					0	1	0	× 1/100	
					0	1	1	× 1/1000	
					1	0	0	× 1/10000	

内部クロック速度 = 20MHz × (A) × (B)

外部クロック速度 = 外部クロック基本速度 × (A)

② 制御ポート II

7	6	5	4	3	2	1	0	機 能	
0								TIME DATA	DO <sub>0</sub> ~ DO <sub>15</sub> の出力制御
1								SAMPLE DATA	
	0							FULL SAMPLING	サンプリング方式の設定
	1							CHANGE SAMPLING	
		0						-TRG	トリガ極性の設定 (立ち上がり/下がり)
		1						+TRG	
			1	×	×	×	×	EXT TRG	外部トリガ入力
			0	0	0	0	0	ch1	内部トリガ入力 チャンネル設定
			0	0	0	0	1	ch2	
			0	0	0	1	0	ch3	
			0	0	0	1	1	ch4	
			0	0	1	0	0	ch5	
			0	0	1	0	1	ch6	
			0	0	1	1	0	ch7	
			0	0	1	1	1	ch8	
			0	1	0	0	0	ch9	
			0	1	0	0	1	ch10	
			0	1	0	1	0	ch11	
			0	1	0	1	1	ch12	
			0	1	1	0	0	ch13	
			0	1	1	0	1	ch14	
			0	1	1	1	0	ch15	
			0	1	1	1	1	ch16	

③ ステータス・ポート

7	6	5	4	3	2	1	0	機 能	
1	1	1	1					TRG ENABLE 前	書き込み時のステータス出力
0	1	1	1					TRG ENABLE 中	
0	0	1	1					TRG ACTIVE 後	
0	0	0	1					TIME OVER測定終了	
0	0	1	0					MEM OVER 測定終了	
						0	0		プローブのステータス出力
						0	1		
						1	0		
						1	1	プローブ無接続	

〈プログラムの説明〉

- プログラムはデータ取り込みルーチンと、データ処理ルーチンに分けられます。
  - 測定ルーチンは、測定条件の設定、測定、測定データの時間換算の3つのブロックに分かれます。
  - PC9801Fぐらいまでは、BASIC領域でのI/Oアドレスは8ビット形式になっており、アドレスの節約という意味から、マシン語レベルからの16ビットI/Oアドレスをアクセスします。
  - マシン語のエリアをどこに置くかは、PC9801各バージョンまたはシステムのセットなどによって異なってきます。
  - BASIC配列をたくさん使うので、セグメントの設定は、マニュアルなどを読んで設定してください。  
[リスト 850行目 DSEG=&H1F000]
  - 取り込んだ16ビット2バイト・データは、取り込みデータと、タイム・カウンタの2バイトで、タイム・カウンタは、本文の手法に基いて、時間換算処理したデータとして、配列に入れます。このデータは、シーケンシャル・ファイルとして記憶できるので、個人的処理に応じたプログラムを作り、ファイルとして取り込めば、バラエティに豊んだ処理が可能です。
  - 表示ルーチンは、画面上に、変化点のみをタイミング・チャート形式にして表示します。よって、変化点間の時間は、カーソルA、B2種類のカーソルを用いて表示します。また、表示ルーチンでは、HELPキー入力によって、リスト装置にコマンド表を出力します。
  - プリンタ出力は、変化点の状態、データHEX表記、カーソルの初めからの各変化点における時間を出力します。
  - この出力ルーチンでは、プリンタの“ビット・イメージの印字”や、フィード量の操作を行うため、各社プリンタに対応させるには改造が必要です。ちなみに、ここでは、MP-80 TYPE 2を想定しています。
- NECモードのプリンタなら、COPYコマンドを用いるのが無難です。

```

10 GOSUB B40
20 DEF SEG=DSEG:CLR ,DSEG
30 GOSUB B40
40 DIM A(1023),B(1023),AA$(16),BB$(16):SCREEN 0,0,0,1
50 HARE=AHIA: DIO CONTROL ADDRESS HIGH BYTE
60 LADRES=AHDO: DIO CONTROL ADDRESS LOW BYTE
70 *****
80 ** LOGIC ANALYSER PROGRAM VOL 1.4 *****
90 ** EPSON PRINTER TYPE **
100 **
110 ** FOR PC-9801F AT 87/03/03 **
120 ** PROGRAM BY T.SAWAI **
130 ** *****
140 **
150 **
160 ** DIO-3298BPC I/O ADDRESS IS DO*D6
170 **
180 ** DIO-3298BPC / JP-2 Δ "L" = 76t
190 **
200 ** PORT
210 **
220 *000*007 --- SAMPLE DATA HBIT/TIME DATA HBIT
230 *
240 *200*207 --- SAMPLE DATA LBIT/TIME DATA LBIT
250 *400*403 --- STI*ST4
260 *404 --- (16) /MEMORY FULL (0)
270 *405 --- (32) /TIMER COUNT FULL (0)
280 *406 --- (64) /TRIGER (0)
290 *407 --- (128) /TRIGER ENABLE (0)
300 *
310 *200*202 --- SAMPLE CLOCK SET A
320 * 200 201 202 N1
330 * (0) 0 0 0 1
340 * (1) 1 0 0 1/10
350 * (2) 0 1 0 1/100
360 * (3) 1 0 1 1/1000
370 * (4) 0 0 1 1/10000
380 *203*204 --- SAMPLE CLOCK SET B
390 * 203 204 N2
400 * (0) 0 0 0 EXT CLOCK
410 * (8) 1 0 1 1/2
420 * (16) 0 1 1 1/4
430 * (24) 1 1 1 1/4
440 *
450 *205 --- NC
460 *206 --- NC
470 *207 --- (128) READ/WRITE (0)
480 *
490 *400*403 --- LINE TRIGER SET
500 * 400 401 402 403
510 * (0) 0 0 0 0 CH1
520 * (1) 1 0 0 0 CH2
530 * (2) 0 1 0 0 CH3
540 * (3) 1 1 0 0 CH4
550 * (4) 0 0 1 0 CH5
560 * (5) 1 0 0 1 CH6
570 * (6) 0 1 1 0 CH7
580 * (7) 1 1 1 0 CH8
590 * (8) 0 0 0 1 CH9
600 * (9) 1 0 0 1 CH10
610 * (10) 0 1 0 1 CH11
620 * (11) 1 1 0 1 CH12
630 * (12) 0 0 1 1 CH13
640 * (13) 1 0 1 1 CH14
650 * (14) 0 1 1 1 CH15
660 * (15) 1 1 1 1 CH16
670 *404 --- EXT/LINE
680 *405 --- *TRG*/TRG (0)
690 *406 --- (64)CHANGING POINT/FULL SAMPLE (0)
700 *407 --- (128) SAMPLE DATA/TIME DATA (0)
710 *
720 *FRMS=0: IF PRMS=1 THEN NEC ELSE EPSON
730 BLDP=61: ---BIT IMAGE /ICHR
740 FFS=CHR$(12): ---BIT IMAGE /ICHR
750 CRS=CHR$(8)&H:
760 LFS=CHR$(8)&H:

```

```

770 F10*=CHR$(8)&H1B)+ "A"+CHR$(8)&H1:
780 F70*=CHR$(8)&H1B)+ "A"+CHR$(8)&H7:
790 F80*=CHR$(8)&H1B)+ "A"+CHR$(8)&H8:
800 L132*=CHR$(8)&HF:
810 L80*=CHR$(8)&H1:
820 B18=CHR$(8)&H1B)+ "K"+CHR$(6)+CHR$(0):
830 GOTO 1350
840 *
850 *-----segment load-----
860 DSEG=&H1F00: PC-9801E,F SEGMENT
870 RETURN IF PC-9801VM THEN DSEG=&H2E00
880 *
890 *-----DIO TABLE-----FOR PC-9801F
900 *
910 *
920 *SETPR2
930 POKE &H1,LADRES+2:POKE &H2,HADRES:POKE &H4,0D
940 ADRES=&H0:CALL ADRES
950 RETURN
960 *SETPR4
970 POKE &H1,LADRES+4:POKE &H2,HADRES:POKE &H4,0D
980 ADRES=&H0:CALL ADRES
990 RETURN
1000 *START
1010 POKE &H1,LADRES+6:POKE &H2,HADRES:POKE &H4,0
1020 ADRES=&H0:CALL ADRES
1030 RETURN
1040 *COUNTUP
1050 POKE &H1,LADRES:POKE &H2,HADRES:POKE &H4,0
1060 ADRES=&H0:CALL ADRES
1070 RETURN
1080 *INP0
1090 POKE &H16,LADRES:POKE &H17,HADRES:ADRES=&H10
1100 CALL ADRES
1110 AH=PEEK(&H20)
1120 RETURN
1130 *INP2
1140 POKE &H16,LADRES+2:POKE &H17,HADRES:ADRES=&H10
1150 CALL ADRES
1160 AL=PEEK(&H20)
1170 RETURN
1180 *INP4
1190 POKE &H16,LADRES+4:POKE &H17,HADRES:ADRES=&H10
1200 CALL ADRES
1210 STUS=PEEK(&H20)
1220 RETURN
1230 *SET, MCH
1240 ADRES=0:RESTORE 1280
1250 READ MDAS:IF MDAS="END" THEN 1320
1260 POKE ADRES,VAL("&H"+MDAS):ADRES=ADRES+1
1270 GOTO 1250
1280 DATA B6,D0,1A,B0,00,EE,CF,90,90,90,90,90,90,90
1290 DATA B8,00,1E,BE,DB,B4,D4,1A,EC,BB,20,00,BB,07,CF,90
1300 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1310 DATA END
1320 POKE &H12,DSEG * 256
1330 POKE &H11,DSEG MOD 256
1340 RETURN
1350 *
1360 *
1370 *
1380 *
1390 *MENU INPUT
1400 *WIDTH 80,25
1410 *CONSOLE 0,25,0,1:CLS 3:COLOR 7
1420 *PRINT "***** LOGIC ANALYZER MODOK1 *****"
1430 *LOCATE 50,2,1:PRINT " / MENU"
1440 *PRINT " 1) DATA SAMPLE"
1450 *PRINT " 2) DATA PRINT (16CH TIMING FORMAT)"
1460 *INPUT "JOB ?":M
1470 *ON M GOSUB 6300,1490
1480 *GOTO 1460
1490 *TIMING CHART PROGRAM
1500 *ON HELP GOSUB 3370
1510 *WIDTH 80,20:CONSOLE 0,20,0,1
1520 *

```



```

1530
1540 IF DIL=1 THEN 1550 ELSE GOSUB 8840
1550 PRINT CHR$(12)
1560 CONSOLE 2,18,0,1:VFA=4:VPB=4
1570 FOR I=1 TO 16:LOCATE 0,I+2:PRINT I:NEXT
1580 LOCATE 0,3,1:PRINT"L";
1590 LOCATE 0,16,1:PRINT"R";
1600 LOCATE 0,1,1:PRINT "TRIGGER INPUT IS (";TRG$;"CH ";TRG$;"TRIGGER)"
1610 DVTI#:=TPT*ST
1620 LOCATE 40,1,1:PRINT "TRIGGER TIME IS ";GOSUB 2870
1630 PRINT "(";ST$;" ";
1640 IF DIL=1 THEN 1780 ELSE 1650
1650 LOCATE 0,1,1:PRINT "
1660 INPUT #1,DS,EST,TPT,ST,ST$,TRG$,TRG$
1670 LOCATE 0,1,1:PRINT "TRIGGER INPUT IS (";TRG$;"CH ";TRG$;"TRIGGER)"
1680 DVTI#:=TPT*ST
1690 LOCATE 40,1,1:PRINT "TRIGGER TIME IS ";GOSUB 2870
1700 PRINT "(";ST$;" ";
1710 I:=1
1720 IF EOF(1) THEN 1770
1730 INPUT #1,A,E
1740 A(I)=A:B(I)=E:I:=I+1
1750 IF E=TPT THEN PT=I-1
1760 GOTO 1720
1770 CLOSE #1
1780 HELP ON
1790 HO=3:PK1=11:PK2=1:PK3=PK1
1800 MAXPAB=(PK3 * 74)+1:FC=70 * MAXPAB
1810 PT1=((PT-(PT * 74)+74)*PC) * 74+(PT * 74)*PC
1820 MEN=((PK3-(PK3 * 74)+74)*PC) * 74+(PK3 * 74)*PC
1830 FOR KK=1 TO I-1
1840 HO=HO+1
1850 A=AKK:B=B(KK)
1860 COLOR 7
1870 IF KK=PK2 THEN COLOR 1
1880 IF KK=PK1 THEN COLOR 6
1890 IF B=TPT THEN COLOR 2
1900 P=A:SD=A:TD=B
1910 SA=SD:FOR I=1 TO 16
1920 SD=INT(SDA/2)
1930 IF 2*SD=SDA THEN AAB=" " ELSE AAB="-"
1940 LOCATE HO,I+2:PRINT AAB
1950 SDA=SD:SD=SD+1
1960 IF B=TPT THEN BEEP
1970 IF KK=PK2 THEN LOCATE HO,19,1:PRINT"B";L=HO+1:VPB=HO
1980 IF KK=PK1 THEN LOCATE HO,19,1:PRINT"A";J=HO+1:VFA=HO
1990 IF B=TPT THEN LOCATE HO,19,1:PRINT"R";
2000 IF HO=77 THEN GOSUB 3040:HO=3
2010 IF KK=I-1 THEN GOSUB 3040:HO=3
2020 NEXT KK
2030 GOSUB 3040:HO=3
2040 GOTO 2030
2050
2060 RETURN 3040
2070 AD=TE VFA,19,1:PRINT " ";
2080 LOCATE (J-1)*8-1,31)-(J-1)*8-1,190),0
2100 J:=4:GOTO 2210
2110
2120
2130
2140 PK1=KK+J-(HO+1):NUM=PK1:GOSUB 3420
2150 DS=INKEY$
2160 IF DS="" THEN 2150
2170 IF ASC(DS)=4H1C THEN 2210
2180 IF ASC(DS)=4H1D THEN 2290
2190 IF DS="S" OR DS="s" THEN RETURN 3040:ELSE 2150
2200 CURSOR RIGHT
2210 J=J+1
2220 VFA=J-1
2230 IF J>HO+1 THEN J=HO+1:VFA=HO
2240
2250 LOCATE VFA-1,19,1:PRINT " ";LOCATE VFA,19,1:PRINT "A";
2260 LOCATE (J-2)*8-1,31)-(J-2)*8-1,190),0
2270 LOCATE (J-1)*8-1,31)-(J-1)*8-1,190),5:GOTO 2350
2280
2280 CURSOR LEFT
2290 J=J-1:IF J<5 THEN J=5
2300 VFA=J-1
2310
2320 LOCATE VFA-1,19,1:PRINT " ";LOCATE VFA,19,1:PRINT "R";
2330 LOCATE (J-1)*8-1,31)-(J-1)*8-1,190),0:LINE (J-1)*8-1,31)-(J-1)*8-1,190),6
2340 GOTO 2340+L-(HO+1):NUM=PK2:GOSUB 3420
2350
2350 IF PT=PK2 THEN LOCATE VFA-1,19,1:PRINT"R";GOTO 2370
2360 IF PT=PK1 THEN LOCATE VFA-1,19,1:PRINT"R";
2370 IF PK2=KK AND PK2=KK-(HO-4) THEN 2380 ELSE 2390
2380 LOCATE VFA,19,1:PRINT "B";LINE (L-1)*8-1,31)-(L-1)*8-1,190),1
2390 DS=INKEY$:IF DS="" THEN 2130 ELSE 2390
2400 B=J-VFA
2410 LOCATE VFA,19,1:PRINT " ";
2420 LOCATE (L-1)*8-1,31)-(L-1)*8-1,190),0:L=4:GOTO 2510
2430
2440 PK2=KK+L-(HO+1):NUM=PK2:GOSUB 3420
2450
2460 DS=INKEY$
2470 IF DS="" THEN 2460
2480 IF ASC(DS)=4H1C THEN 2510
2490 IF ASC(DS)=4H1D THEN 2590
2500 IF DS="S" OR DS="s" THEN RETURN 3040 ELSE 2460
2510 L=L+1
2520 VFA=L-1
2530 IF L>HO THEN L=HO+1:VFA=HO
2540
2550 LOCATE VFA-1,19,1:PRINT " ";LOCATE VFA,19,1:PRINT "B";
2560 LOCATE (L-2)*8-1,31)-(L-2)*8-1,190),0
2570 LOCATE (L-1)*8-1,31)-(L-1)*8-1,190),1
2580 GOTO 2660
2590 L=L-1:IF L<5 THEN L=5
2600 VFA=L-1
2610
2620 LOCATE VFA-1,19,1:PRINT " ";LOCATE VFA,19,1:PRINT "B";
2630 LOCATE (L)*8-1,31)-(L)*8-1,190),0
2640 LOCATE (L-1)*8-1,31)-(L-1)*8-1,190),1
2650 GOTO 2670
2660
2670 IF PT=PK2 THEN LOCATE VFA-1,19,1:PRINT "R";GOTO 2680
2680 IF PT=PK1 AND PK1=KK-(HO-4) THEN 2690 ELSE 2710
2690 LOCATE VFA,19,1:PRINT "A";
2700 LOCATE (J-1)*8-1,31)-(J-1)*8-1,190),6
2710 DS=INKEY$:IF DS="" THEN 2440 ELSE 2710
2720
2730 DVTI#=(B(PK1)-B(PK2))*ST:GOSUB 8520
2740 PRDATA$="A"B="+STR$(DVTI#)+DVTI$
2750 RETURN
2760
2770 DVTI#=(B(PK1)-B(PK2))*ST:GOSUB 8520
2780 PRDATA$="B"A="+STR$(DVTI#)+DVTI$
2790 RETURN
2800
2810 RETURN 3040
2820 CLS
2830 FOR I=1 TO 16:LOCATE 0,I+2:PRINT I:NEXT
2840 LOCATE 0,3,1:PRINT"L";
2850 LOCATE 0,16,1:PRINT"R";
2860 RETURN
2870 IF DVTI#>1000000000 THEN DVTI#:=DVTI#/1000000000:KN=4:GOTO 2910
2880 IF DVTI#>1E+06 THEN DVTI#:=DVTI#/1E+06:KN=3:GOTO 2910
2890 IF DVTI#>1000 THEN DVTI#:=DVTI#/1000:KN=2:GOTO 2910
2900 KN=1
2910 PRINT USING "#####";DVTI#;
2920 IF DS="EXT " THEN 2930 ELSE 2980
2930 ON KN GOTO 2940,2950,2960,2970
2940 PRINT " ";GOTO 3030
2950 PRINT "E+3":GOTO 3030
2960 PRINT "E+6":GOTO 3030
2970 PRINT "E+9":GOTO 3030
2980 ON KN GOTO 2990,3000,3010,3020
2990 PRINT " nS":GOTO 3030
3000 PRINT " uS":GOTO 3030
3010 PRINT " mS":GOTO 3030
3020 PRINT " S";
3030 RETURN

```

```

3040 CP=KK:CFEN=CP1:CFST=CP-(HO-3)
3050 CU1=(CFST&PC)74
3060 CUEN=(CFEN&PC)74
3070 GOSUB 3540
3080 LOCATE 0,2,1:PRINT " "
3090 LOCATE 0,2,1:COLOR 3:INPUT "COMMAND ";CM$
3100 LOCATE 0,2,1:PRINT " "
3110 COLOR 7
3120 GOSUB 3690
3130 IF LEFT$(CM$,2)="LP" OR LEFT$(CM$,2)="LP" THEN GOSUB 4900:GOTO 2820
3140 IF LEFT$(CM$,2)="PA" OR LEFT$(CM$,2)="pa" THEN GOTO 3160
3150 KK=PK1:HO=3:GOSUB 2820:RETURN 1840
3160 IF LEFT$(CM$,2)="PB" OR LEFT$(CM$,2)="pb" THEN 3170 ELSE 3180
3170 KK=PK2:HO=3:GOSUB 2820:RETURN 1840
3180 IF LEFT$(CM$,2)="PT" OR LEFT$(CM$,2)="pt" THEN 3190 ELSE 3200
3190 KK=PT:HO=3:GOSUB 2820:RETURN 1840
3200 IF COSEL=2 THEN 3210 ELSE 3230
3210 IF KK=74:PNUM#75:PK3 THEN KK=PK3-73:GOSUB 2820:HO=3:RETURN 1840
3220 KK=KK-74:PNUM#75:GOSUB 2820:HO=3:RETURN 1840
3230 IF LEFT$(CM$,3)="CSA" OR LEFT$(CM$,3)="csa" THEN GOSUB 2400
3240 IF LEFT$(CM$,3)="CSB" OR LEFT$(CM$,3)="csb" THEN GOSUB 2070
3250 IF COSEL=1 THEN 3260 ELSE 3280
3260 IF KK=74:PNUM#75:1 THEN KK=1:GOSUB 2820:HO=3:RETURN 1840
3270 KK=KK-74:PNUM#75:GOSUB 2820:HO=3:RETURN 1840
3280 IF LEFT$(CM$,1)="E" OR LEFT$(CM$,1)="e" THEN 3360
3290 IF LEFT$(CM$,3)="RTT" OR LEFT$(CM$,3)="rtt" THEN PR2=PT:GOTO 4430
3300 IF LEFT$(CM$,3)="RTA" OR LEFT$(CM$,3)="rta" THEN PR2=PK1:GOTO 4430
3310 IF LEFT$(CM$,3)="RTB" OR LEFT$(CM$,3)="rtb" THEN PR2=PK2:GOTO 4430
3320 IF LEFT$(CM$,2)="RT" OR LEFT$(CM$,2)="rt" THEN PR2=1:GOTO 4430
3330 LOCATE 0,2,1:COLOR 2:PRINT "COMMAND ERROR !!!!!!!!!BEEP FOR 11=1 TO 1000:NEXT
3340 LOCATE 0,2,1:PRINT " "
3350 RETURN
3360 HELP OFF:CLS 3:RETURN 1390
3370 LOCATE 0,2,1:PRINT " "
3380 LOCATE 0,2,1:INPUT "COMMAND MENU PRINT OUT ";Y$
3390 IF Y$="Y" OR Y$="y" THEN 3690
3400 RETURN
3410 PRNDAT$="":PRNDAT$=DATA(HEX)*"RIGHTS("0000"+HEX$(A(NUM)),4)*"
3420 LOCATE 0,2,1:COLOR 7:PRINT PRNDAT$;
3430 LOCATE 0,2,1:COLOR 7:PRINT PRNDAT$;
3440 PRNDAT$="":PRNDAT$=DATA(HEX)*"STR$(DYTI#)+DYTI$
3450 PRNDAT$="":PRNDAT$=DATA(HEX)*"STR$(DYTI#)+DYTI$
3460 LOCATE 0,2,1:COLOR 6:PRINT LEFT$(PRNDAT$+"
3470 PRNDAT$="":PRNDAT$=DATA(HEX)*"STR$(DYTI#)+DYTI$
3480 PRNDAT$="":PRNDAT$=DATA(HEX)*"STR$(DYTI#)+DYTI$
3490 LOCATE 34,2,1:COLOR 1:PRINT LEFT$(PRNDAT$+"
3500 PRNDAT$="":PRNDAT$=DATA(HEX)*"STR$(DYTI#)+DYTI$
3510 IF PK1>PK2 THEN GOSUB 2730 ELSE GOSUB 2770
3520 LOCATE 52,2,1:COLOR 7:PRINT LEFT$(PRNDAT$+"
3530 RETURN
3540 "MAPING SUB
3550 PK1=PK1 74:PK2=PK2 74
3560 PC1=((PK1-PK1&174)&PC) 74:PKK1&PC
3570 PC2=((PK2-PK2&174)&PC) 74:PKK2&PC
3580 PPO$=" "
3590 FOR PCO=0 TO 69
3600 PCO$=" "
3610 IF PCO>XJUST AND PCO<CUEN THEN PCO$=" "
3620 IF MEN<PCO THEN PCO$=" "
3630 IF PT1=PCO THEN PCO$="A"
3640 IF PC1=PCO THEN PCO$="T"
3650 IF PC2=PCO THEN PCO$="B"
3660 PPO$=PPO$+PCO$
3670 NEXT
3680 LOCATE 8,0,1:PRINT PPO$:RETURN
3690 "HELP MENU
3700 PRINT "*****
3710 PRINT " LOGIC ANALYZER
3720 PRINT "*****
3730 SP3=60:GOSUB 5920
3740 PRINT
3750 PRINT "1) "+n$"......DISPLAY PAGE +n COMMAND
3760 PRINT
3770 PRINT "2) "-n$"......DISPLAY PAGE -n COMMAND

```



[illegible]

137



```

7490 PRINT "
7500 PRINT "
7510 PRINT "
7520 PRINT "
7530 PRINT "
7540 PRINT "
7550 PRINT "
7560 PRINT "
7570 PRINT "
7580 PRINT "
7590 PRINT "
7600 PRINT "
7610 PRINT "
7620 PRINT "
7630 PRINT "
7640 PRINT "
7650 LOCATE 55,10:INPUT "SAMPLE TIME (NO) " :M
7660 ON M GOTO 7680,7690,7700,7710,7720,7730,7740,7750,7760,7770,7780,7790,7800,
7810,7820,7830
7670 GOTO 7650
7680 Q2=8:ST=50 nS:ST=50:GOTO 7840
7690 Q2=16:ST=100nS:ST=100:GOTO 7840
7700 Q2=24:ST=200nS:ST=200:GOTO 7840
7710 Q2=8+1:ST=500nS:ST=500:GOTO 7840
7720 Q2=16+1:ST=1 uS:ST=1000:GOTO 7840
7730 Q2=24+1:ST=2 uS:ST=2000:GOTO 7840
7740 Q2=8+2:ST=5 uS:ST=5000:GOTO 7840
7750 Q2=16+2:ST=10 uS:ST=10000:GOTO 7840
7760 Q2=24+2:ST=20 uS:ST=20000:GOTO 7840
7770 Q2=8+3:ST=50 uS:ST=50000:GOTO 7840
7780 Q2=16+3:ST=100uS:ST=100000:GOTO 7840
7790 Q2=24+3:ST=200uS:ST=200000:GOTO 7840
7800 Q2=8+4:ST=500uS:ST=500000:GOTO 7840
7810 Q2=16+4:ST=1 mS:ST=1E+06:GOTO 7840
7820 Q2=24+4:ST=2 mS:ST=2E+06:GOTO 7840
7830 Q2=0:ST="EXT " :ST=1:GOTO 7840
7840 RETURN
7850 PRINT CHR$(12)
7860 LOCATE 10,16:PRINT "+"
7870 LOCATE 10,19:PRINT "
7880 LOCATE 55,18:PRINT "TRIGGER SLOPE " :
7890 LOCATE 55,19:INPUT " (+/-) " :Y$
7900 IF Y$="+" OR Y$="-" THEN 7920
7910 GOTO 7850
7920 IF Y$="+" THEN Q43=32:TRGP$="+" :GOTO 7940
7930 Q43=0:TRGP$="-"
7940 RETURN
7950 PRINT CHR$(12)
7960 LOCATE 10,15:INPUT "CHANGE POINT SAMPLE (Y/N) " :Y$
7970 IF Y$="Y" OR Y$="y" THEN Q44=64:DS=1:GOTO 7990
7980 Q44=0:DS=0
7990 RETURN
8000 PRINT CHR$(12)
8010 PRINT "
8020 PRINT "
8030 PRINT "
8040 PRINT "
8050 PRINT "
8060 PRINT "
8070 PRINT "
8080 PRINT "
8090 PRINT "
8100 PRINT "
8110 PRINT "
8120 PRINT "
8130 PRINT "
8140 PRINT "
8150 LOCATE 55,10:PRINT "TRIGGER INPUT CHANNEL "
8160 LOCATE 55,11:INPUT " (1~17) ? " :N
8170 IF N<0 OR N>16 THEN 8000
8180 IF N=16 THEN 8000
8190 Q42=16:TRG$="EXT" :RETURN
8200 Q42=N:TRG$=STR$(N+1)+" "
8210 TRG$=LEFT$(TRG$,3)
8220 RETURN
8230 ES=0:TIME$="00:00:00"
8240 GOSUB $INP4

```

```

8250 SH$=LEFT$(RIGHT$(%0"+HEX$(STUS),2),1)
8260 IF SH$="F" THEN LOCATE 25,15,1:PRINT " START
8270 IF SH$="7" THEN LOCATE 25,15,1:PRINT "TRIGGER ENABLE "
8280 IF SH$="3" THEN LOCATE 25,15,1:PRINT "GOTO TRIGGER "
8290 IF SH$="1" AND DS=1 THEN LOCATE 25,15,1:PRINT "TIMER COUNT FULL" :ES=1
8300 IF SH$="2" OR SH$="0" THEN LOCATE 25,15,1:PRINT "MEMORY FULL " :ES=2
8310 IF (SH$="2" OR SH$="0") AND SH$="0" OR SH$="0" AND ES<>0 THEN RETURN
8320 GOTO 8240
8330 "SAMPLE DATA LINE PRINT
8340 IF DS=1 THEN DS$="CHANGE" ELSE DS$="FULL "
8350 SW$=LEFT$(TRG$,4)
8360 DYTI#="TPT&ST:GOSUB 8520
8370 FIL$=LEFT$(FIL$,5)
8380 LPRINT F7D$:
8390 LPRINT "
8400 LPRINT " : SAMPLING SPECIFICATION FILE NAME " :FIL$: "
8410 LPRINT "
8420 LPRINT " : SAMPLING FORMAT " :DS$: "
8430 LPRINT "
8440 LPRINT " : TRIGGER INPUT " :SW$: "
8450 LPRINT " : TRIGGER SLOPE " :TRGP$: "TRIGR
"
8460 LPRINT USING " : TRIGGER POINT (START?) " :
8470 LPRINT DYTI$: "
8480 LPRINT "
8490 LPRINT "
8500 LPRINT F8D$:
8510 RETURN
8520 IF DYTI#>1000000000000 THEN DYTI#="DYTI#/1000000000000:KN=4:GOTO 8560
8530 IF DYTI#>1E+06 THEN DYTI#="DYTI#/1E+06:KN=3:GOTO 8560
8540 IF DYTI#>1000 THEN DYTI#="DYTI#/1000:KN=2:GOTO 8560
8550 KN=1
8560 IF ST$="EXT " THEN 8570 ELSE 8620
8570 ON KN GOTO 8580,8590,8600,8610
8580 DYTI$="E3":GOTO 8670
8590 DYTI$="E3":GOTO 8670
8600 DYTI$="E6":GOTO 8670
8610 DYTI$="E6":GOTO 8670
8620 ON KN GOTO 8630,8640,8650,8660
8630 DYTI$="NS":GOTO 8670
8640 DYTI$="NS":GOTO 8670
8650 DYTI$="MS":GOTO 8670
8660 DYTI$="MS":GOTO 8670
8670 RETURN
8680 RETURN
8690 COSEL=0
8700 FOR SLIS=1 TO LEN(CM$)
8710 SLIS=MID$(CM$,SLIS,1)
8720 IF COSEL=0 THEN GOSUB 8800
8730 IF COSEL>0 THEN 8740 ELSE 8740
8740 IF ASC(SLIS)>=48&0 AND ASC(SLIS)<=57&0 THEN PNUM$=PNUM$+SLIS$
8750 IF SLIS$="p" OR SLIS$="P" THEN 8770
8760 NEXT COSEL=0:GOTO 8790
8770 PNUM=VAL(PNUM$)
8780 IF PNUM=0 THEN PNUM=1
8790 RETURN
8800 IF SLIS$="n" THEN COSEL=1:PNUM$="":RETURN
8810 IF SLIS$="x" THEN COSEL=2:PNUM$="":RETURN
8820 COSEL=2:PNUM$="":RETURN
8830 DUMY=B2+1
8850 PRINT CHR$(12)
8860 PRINT "
8870 PRINT "
8880 PRINT "
8890 INPUT "FILE NAME ? INPUT 2? 7? 7? " :FIL$
8900 IF FIL$="FILES" OR FIL$="files" THEN GOTO 8990
8910 IF FIL$="FILES 1" OR FIL$="files 1" THEN GOTO 8990
8920 IF FIL$="FILES 2" OR FIL$="files 2" THEN GOTO 9000
8930 IF LEFT$(FIL$,5)="FILES" OR LEFT$(FIL$,5)="files" THEN 8850
8940 IF FIL$=" " THEN FIL$="7?7?"
8950 OPEN FIL$ AS #1
8960 IF EOF(1)=1 THEN 8970 ELSE RETURN
8970 CLOSE:PRINT "NO file "
8980 KILL FIL$:GOTO 8850
8990 PRINT CHR$(12):FILES:PRINT :GOTO 8860
9000 PRINT CHR$(12):FILES 2:PRINT :GOTO 8860

```

# PC9801のグラフLIOの詳細と活用法

本章では、PC9801の内部コマンドであるLIOの使い方の例としてグラフLIOを取り上げ、その詳細と活用法を解説します。

パソコンも8ビット・マシンの初期の頃は、外部記憶装置としてはカセット・テープレコーダが中心で、その他のハードウェアも、よく知られた汎用LSIが使われ、ユーザがこれらを制御するプログラムを作成する際のソフトは、BASIC、あるいはハンド・アセンブルによる16進入力などで十分に間に合うものでした。

これに対して、昨今になって著しい普及をみせている16ビット・パソコンのハードウェアは、**強力なグラフィックス処理機能や漢字処理機能**をもち、また、OS(Operating System)走行環境のためのディスクなども標準で装備されています。これに伴いGDC(Graphic Display Controller)やFDC(Floppy Disk Controller)など、高機能なコントローラが多く用いられているので、ユーザがこれらのコントローラ(LSI)のコマンドなどを理解し、その制御プログラムまで作成して、パソコンのもつ高機能なハードウェアを自由に使いこなすのは容易なことではありません。

しかし幸いなことに、PC9801シリーズなどでは、**ユーザがソフトウェア開発(BASIC以外)の際に利用できるように、メーカ側がこれらの基本的な制御プログラム(BIOやLIO)をROM内に収めて供給**しており、ユーザは必要に応じてこれらのサブルーチン群をコールすることにより、高機能で複雑なI/O制御も簡単に行うことができるように設計されています。

ここでは、それらの活用の一例として、C言語から利用できるグラフィック・ライブラリを紹介します。

## □PC9801のソフトウェア構成とBIOS

### 1-1 ソフトウェアの構造

BIOSとはハードウェアを直接制御する**制御プログラムの集まり**で、この中にはグラフィック関係のBIO、ディスク関係のBIO、さらにはマウスやRS-232C、ひいてはGPIO、ミュージック・ジェネレータなどのBIOが含まれています。

そして、このBIOを介してより論理的なI/Oアクセスを行うためのLIOがあります。これもグラフィック

関係やディスク関係、あるいはプリンタ関係など、各種のLIOが含まれていて、このLIOを利用する際には、BASICコマンドと類似のパラメータの受け渡しが行われます。

さらに、その上にシステム用のソフト(BASICやDOSなど)があり、これらのBIOやLIOを介してI/Oアクセスを行っています。

このBIOやLIOの中には有用なサービス・ルーチンが数多く含まれ、またROM内にはBIOやLIOだけでなく、**実数演算や浮動小数点演算ルーチンなどもROM化**されていて、アセンブリ言語などからも利用できるようになっています。

### 1-2 ブート・ストラップ動作

まず、電源スイッチを入れた後の動作、つまりブート・ストラップ動作について説明しておきましょう。

PC9801シリーズでは、メディアのタイプに320KバイトFD(5"/2D)、640KバイトFD(3.5", 5"/2DD)、1MバイトFD(3.5", 5", 8"), 5"ハード・ディスクなどがあり、またDOSシステムもN<sub>86</sub>BASIC、CP/M86、MS-DOSなどが供給されています。ただし、ブート手順はほとんど変わりありません。

電源スイッチを入ると、まず、ブート・ロードが呼ばれます。このブート・ロードでは、ROM内のBIOを介してディスクからIPL(Initial Program Loader: 0シリンダ0トラックに書かれている。セクタ・サイズなどは、DOSやメディアの種類によって異なる)を指定したアドレス(単密度の場合1F E 0 0 hから1F F F F F hまでの512バイト、倍密度の場合1F C 0 0 hから1F F F F F hまでの1Kバイト)にロードし、そのIPLに制御を移します。

このブート・ロードにもいくつかの種類があり、それぞれ入力条件が異なりますが、次の点ではすべての方法で共通しています。

- ① すべてのレジスタは保存されない
- ② ブート動作が失敗したらCALL ERにもどる
- ③ ブート動作が正常に終了すればIPLにジャンプする



〈表1-1〉 ブート・プライオリティ

プライオリティ	対 応 装 置
1	1MバイトFD (VM)
2	640KバイトFD
3	
4	1MバイトFD (8")
5	
6	
7	
8	320KバイトFD (5"2D)
9	
10	5"HD (UNIT0)
11	5"HD (UNIT1)
12	
13	
14	
15	

### ▶ ブート方法①

これは、メモリ・スイッチSW<sub>8</sub>(マニュアル参照)の指定にしたがった、システムの立ち上げを行う場合に使用します。

CALL条件

DS←0000h

CALL FD80:27ECh

### ▶ ブート方法②

ブート装置をプライオリティ順序(表1-1)から選択して、システムの立ち上げを行う場合に使用します。

CALL条件

DS←0000h

AL←プライオリティ初期値

AH←プライオリティ終了値

CALL FD80:27E8h

### ▶ ブート方法③

拡張ROM内のブート・ロードを直接呼び出す場合に使用します。

CALL条件

DS←0000h

AL←02h (640KバイトFD)

04h (1MバイトFD)

CALL D600:0015h (640KバイトFD)

CALL D700:0015h (1MバイトFD)

## 1-3 ディスクBIO

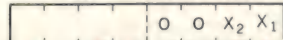
BIOSの例として、最も利用頻度が高いと思われるディスクBIOについて触れておきましょう。

ディスク関係のハードウェア(FDCなど)を直接制御するソフトウェアとして、ディスクBIOがあります。このディスクBIOを用いると、例えばフォーマット・コマンドやファイル・コンバータなどの作成において、

〈図1-1〉 DISK BIOS(INT 1Bh)の入出力パラメータ

入力パラメータ	出力パラメータ
AH←BIOSコマンド識別コード	CF(フラグ)←終了条件(0:正常,1:異常)
AL←デバイス種別・ユニット番号	AH←終了ステータス
BX←データ長(バイト長)	CL←シリンダ番号
CL←シリンダ番号(0~76)	CH←セクタ長
CH←セクタ長(0~3)	DL←セクタ番号
DL←セクタ番号(1~26)	DH←ヘッド番号
DH←ヘッド番号(0~1)	

デバイス種別・ユニット番号( ALレジスタ)



ユニット(ドライブ)番号(0~3)

インターフェース種別

{000: 5"HD  
001: 1Mバイト・インターフェース  
111: 640Kバイト・インターフェース

アクセス・モード

{0: 640Kバイト・アクセス  
1: 1Mバイト・アクセス

〈図1-2〉 BIOS識別コード(AHレジスタ)

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
M	T	F	S	E	E	K	コマン
T	F		E	E	E	K	ド・コード

ビット	機能	動 作	コマンドコード(D <sub>1</sub> ~D <sub>0</sub> )	動 作
D <sub>7</sub>	MT	0: シングル・トラック 1: マルチ・トラック	0001	ベリファイ
D <sub>6</sub>	MF	0: 単 密 度 1: 倍 密 度	0101	データ書き込み
D <sub>5</sub>	F	0: 8回リトライする 1: 8回リトライしない	0110	データ読み出し
D <sub>4</sub>	SEEK	0: SEEKしない 1: SEEKする	1010	ID読み出し
			1101	ID書き込み

FDCやDMACの詳細について知らなくても、簡単に目的のプログラムが作成できます。

このディスクBIOの割り込み番号は、フロッピー・ディスクが1Bh、ハード・ディスクがB1hとなっています。

ここでは、フロッピー・ディスク用のディスクBIO(INT 1Bh)について、その入出力パラメータを中心に解説します。

ディスクBIOをコールする際には、図1-1に示した入力パラメータをセットしなければなりません。

AHレジスタにセットすべきBIOSコマンド識別コードとは、図1-2のような構成になっており、下位4ビットで読み出しや書き込みなどのコマンドの種類、上位4ビットでディスク・アクセスに関する細かい指定が行われます。

また、ALレジスタにセットするデバイス種別/ユニット番号により、ディスク・メディアの種類やドライブ番号を指定します。

CHレジスタにはセクタ長フォーマットを表1-2のようにセットします。

これらのパラメータをセットして

〈表1-2〉セクタ長フォーマット

CH レジスタ値	バイト/セクタ
00h	128
01h	256
02h	512
03h	1024

〈表1-3〉  
AHレジスタに返される  
出力ステータス

CF	AH	内 容
0	00h	正常終了
	10h	ライト・プロテクト (senceコマンド)
	20h	DMA 不可
	30h	転送容量超過
	40h	デバイス異常
	50h	データ転送が間に合わない
	60h	ユニットのノット・レディ
1	70h	ライト・プロテクト
	80h	エラー
	90h	タイム・アウト
	A0h	CRC エラー
	B0h	ID データ・エラー
	C0h	DATA なし
	D0h	シリンドラ設定不良
	E0h	ID アドレス・マーク・エラー
	F0h	DATA アドレス・マーク・エラー

## INT 1Bh

を実行すると、図1-1のような出力パラメータが返され、キャリ・フラグ(C)が“0”の場合は正常終了になります。キャリ・フラグが“1”の場合はエラーであり、そのエラー・ステータスはAHレジスタに返され、その内容は表1-3のようになっています。

この“INT 1Bh”ルーチンは、ファイル・コンバータやコピー・プロテクトなど、幅広い応用が考えられます。

## ②グラフィックに関するLIOのサポート

### 2-1 GDCについて

PC9801シリーズでは、CRTインターフェースにGDC(μPD7220)が2個用いられており、そのうちの1個はテキスト画面用にマスタ動作で用いられ、残りの1個はスレーブ動作でグラフィック画面に用いられています。

PC9801シリーズでは、このように高機能なGDCを

用いたことにより、CRT関係の処理をGDCが行いますから、CPUの負担が軽減され、システム全体のスループットが向上しています。

ここでは、このGDC(μPD7220)について簡単に触れておくことにします。GDCではCPUとのインターフェースに、コマンド・レジスタ(テキスト用：I/Oアドレス62h、グラフィック用：A2h)とパラメータ用のFIFOメモリ(テキスト用：60h、グラフィック用：A0h)の二つのレジスタが用意されています。

そして、実行コマンドをコマンド・レジスタに書き込み、そのコマンド実行に必要なパラメータをFIFO

〈図2-1〉  
GDCステータス・  
レジスタ

b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>
LIGHT PEN DETECT	HORIZONTAL SYNC	VERTICAL SYNC	DMA EXECUTIVE	DRAWING	FIFO EMPTY	FIFO FULL	DATA READY

〈表2-1〉GDCのコマンド/パラメータ(動作制御)

コマンド名	機 能	C/P	コマンドまたはパラメータ・コード								パラメータ解説
			b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	
RESET	初期化	C	0	0	0	0	0	0	0	0	
SYNC	動作モード、同期タイミングの定義	C	0	0	0	0	1	1	1	DE	DE=0:表示停止、DE=1:表示開始
		P <sub>1</sub>	0	0	CHR	F	I	D	G	S	CHR=1:文字モード、F=1:フラッシュレス描画 I=1:インターレース・モード、D=1:ダイナミックRAM G=1:グラフィック・モード、S=1:インターレース・シュリンク・モード(I=1)
		P <sub>2</sub>	C/R								C/R: 1行の表示文字数
		P <sub>3</sub>	VS <sub>L</sub>		HS					VS: 水平同期期間 HS: 垂直同期期間	
		P <sub>4</sub>	HFP				VS <sub>H</sub>				HFP: 水平右側非表示期間 VS <sub>H</sub> : 垂直上側非表示期間
		P <sub>5</sub>	HBP								HBP: 水平左側非表示期間
		P <sub>6</sub>	VFP								VFP: 垂直上側非表示期間
		P <sub>7</sub>	L/F <sub>L</sub>								L/F: 1画面の表示ライン数
		P <sub>8</sub>	VBP				-L/F <sub>H</sub>				
MASTER/ SLAVE	マスタ動作、スレーブ動作の選択	C	0	1	1	0	1	1	1	M	M=1:マスタ動作、M=0:スレーブ動作



〈表2-2〉 GDCのコマンド/パラメータ(表示制御)

コマンド名	機 能	C/P	コマンドまたはパラメータ・コード								パラメータ解説	
			b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>		
START	表示開始	C	0	1	1	0	1	0	1	1	} どちらでもよい	
STOP	表示停止	C	0	0	0	0	1	1	0	1		
ZOOM	拡大係数の設定	C	0	1	0	0	0	1	1	0	ZR: 拡大表示時の拡大係数 ZW: 拡大描画時の拡大係数	
		P	← ZR →				← ZW →					
SCROLL	表示開始アドレス、表示領域の設定 画面を $n$ 個に分割した場合、このパラメータを $n$ 回送出。 文字モード: $n_{max}=4$ 文字/グラフィック・モードと $n_{max}=2$ グラフィック・モード	C	0	1	1	1	← RA →					RA: データ RAM のアドレス
		P <sub>n1</sub>	← SAD <sub>nL</sub> →								SAD <sub>n</sub> : $n$ 番目の区間の開始アドレス	
		P <sub>n2</sub>	0	0	0	← SAD <sub>nH</sub> →					SL <sub>n</sub> : $n$ 番目の区間の長さ	
		P <sub>n3</sub>	← SL <sub>nL</sub> →				0	0	0	0	IM: 文字/グラフィック・モード時 IM = 0... 文字表示領域 (その他のモード) IM = 1... グラフィック表示領域 IM = 0	
		P <sub>n4</sub>	*	← IM →				← SL <sub>nH</sub> →				
CSRFORM	カーソル形状などの指定	C	0	1	0	0	1	0	1	1	L/R: 1 行中の表示ライン数(グラフィック・モード時...1) CS=1: カーソル表示あり	
		P <sub>1</sub>	CS	0	0	← L/R →					BD=1: プリンキングなし(CS=1なら常時点灯)	
		P <sub>2</sub>	← BL <sub>L</sub> →		BD	← CST →					CST: カーソル表示開始ライン値	
		P <sub>3</sub>	← CFI →				← BL <sub>H</sub> →				BL: カーソル点滅周期 CFI: カーソル表示終了ライン値	
PITCH	映像メモリの水平方向ワード数の設定	C	0	1	0	0	0	1	1	1	水平方向のワード(16ビット)数	
		P	← P →									
LPEN	ライトペン・アドレスの検出	C	1	1	0	0	0	0	0	0	LAD: ライトペン・アドレス	
	読み出し専用	P <sub>1</sub>	← LAD <sub>L</sub> →									
		P <sub>2</sub>	← LAD <sub>M</sub> →									
		P <sub>3</sub>	← LAD <sub>H</sub> →									
VECTW	描画に必要なパラメータの設定	C	0	1	0	0	1	1	0	0	L=1: 直線描画 T=1: 傾斜しないグラフィックス文字描画 C=1: 円および円弧の描画 R=1: 四辺形描画 SL=1: 傾斜したグラフィックス文字(T=1) DGD=1: グラフィック描画(文字/グラフィック・モード)	
		P <sub>1</sub>	SL	R	C	T	L	← DIR →				
		P <sub>2</sub>	← DC <sub>L</sub> →									
	P <sub>3</sub>	0	DGD	← DC <sub>H</sub> →								
	同様の順序で D <sub>2</sub> , D <sub>1</sub> , DM を送出	P <sub>4</sub>	← D <sub>L</sub> →									
		P <sub>5</sub>	← D <sub>H</sub> →									
VECTE	グラフィックス描画開始	C	0	1	1	0	1	1	0	0		
TEXTW	グラフィックまたはテキスト・コード設定	C	0	1	1	1	1	← RA →				RA: データを書き始めるラスト・アドレス PTN: グラフィック描画時の線のビット・パターン TX <sub>n</sub> : ドット構成データ
		P <sub>1</sub>	← TX <sub>8</sub> またはPTN <sub>L</sub> →									
		P <sub>2</sub>	← TX <sub>7</sub> またはPTN <sub>H</sub> →									
		⋮	⋮									
		P <sub>8</sub>	← TX <sub>1</sub> →									
TEXTE	グラフィック/文字描画の実行開始	C	0	1	1	0	1	0	0	0		
CSRW	描画アドレス設定 文字描画時...P <sub>1</sub> , P <sub>2</sub> のみ 文字モード時...EAD <sub>M</sub> の上位3ビット 0 文字/グラフィック・モード EAD <sub>H</sub> =0	C	0	1	0	0	1	0	0	1	EAD: 描画開始ワード・アドレス dAD: 描画開始ドット・アドレス	
		P <sub>1</sub>	← EAD <sub>L</sub> →									
		P <sub>2</sub>	← EAD <sub>M</sub> →									
		P <sub>3</sub>	← dAD →				0	0	← EAD <sub>H</sub> →			
CSRR	描画アドレス読み出し  読み出し専用	C	1	1	1	0	0	0	0	0	CSRW コマンドに同じ	
		P <sub>1</sub>	← EAD <sub>L</sub> →									
		P <sub>2</sub>	← EAD <sub>M</sub> →									
		P <sub>3</sub>	← EAD <sub>H</sub> →									
		P <sub>4</sub>	← dAD <sub>L</sub> →									
		P <sub>5</sub>	← dAD <sub>H</sub> →									
MASK	マスク・レジスタ値の設定	C	0	1	0	0	1	0	1	0	MASK: マスキング/ドット・アドレス・レジスタ値	
		P <sub>1</sub>	← MASK <sub>L</sub> →									
		P <sub>2</sub>	← MASK <sub>H</sub> →									

〈表2-3〉 GDCのコマンド/パラメータ(映像メモリ制御)

コマンド名	機 能	C/P	コマンドまたはパラメータ・コード								パラメータ解説
			b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	
WRITE	ドット修正モード設定 データの書き込み 繰り返し送出可 バイト転送時はCODE <sub>L</sub> のみでよい	C	0	0	1	WLH	0	MOD			WLH=00:ワード転送 MOD=00:REPLACE =01:未定 =01:COMPLEMENT =10:下位バイト転送 =10:CLEAR =11:上位バイト転送 =11:SET
		P <sub>1</sub>	CODE <sub>L</sub>								
		P <sub>2</sub>	CODE <sub>H</sub>								CODE:V-RAM 書き込みデータ
READ	映像メモリの読み出し	C	1	0	1	WLH	0	MOD			WRITE コマンドに同じ
DMAW	映像メモリへのDMA転送 指示	C	0	0	1	WLH	1	MOD			WRITE コマンドに同じ
DMAR	映像メモリからのDMA転 送指示	C	1	0	1	WLH	1	MOD			WRITE コマンドに同じ

〈リスト2-1〉 GDCのテスト・プログラム

```

1:
2: /* GDC テストプログラム
3:
4: #define STS_REG 0xa0
5: #define CMD_REG 0xa2
6:
7: main()
8: {
9:     char buff[10];
10:    int i,x0,x1,y1,x2,y2,bank;
11:    unsigned int ead;
12:
13:    puts("V033#V033= "); /* テキスト画面の消去 */
14:    loop:
15:        printf("x1,y1(0-640,0-400)= ");
16:        scanf("%d %d",&x1,&y1); /* 開始点 x1,y1 */
17:        printf("x2,y2(0-640,0-400)= ");
18:        scanf("%d %d",&x2,&y2); /* 終了点 x2,y2 */
19:        printf("color(0-2)= ");
20:        scanf("%d",&bank); /* G-RAM バンク */
21:
22:        for (i=0;i<8;i++) /* パターンのセット */
23:            buff[i]=0xff;
24:        gdc(0x78.8,buff); /* textw */
25:        gdc(0x23.0,buff); /* write */
26:
27:        ead=0x4000+bank*0x4000+(int)(x1/16)+40*y1;
28:        buff[0]=ead*0x100; /* 描画開始点 */
29:        buff[1]=ead/0x100;
30:        buff[2]=x1;
31:        gdc(0x49.3,buff); /* csrw */
32:
33:        buff[0]=0x10;
34:        buff[1]=x2*0x100;
35:        buff[2]=x2/0x100;
36:        buff[3]=y2*0x100;
37:        buff[4]=y2/0x100;
38:        gdc(0x4c.5,buff); /* vectw */
39:        gdc(0x68.0,buff); /* texte */
40:        goto loop;
41:    }
42:
43:    gdc(com,max,buff) /* GDCにコマンド,パラメータを送る */
44:    int max,com;
45:    char *buff;
46:    {
47:        int i;
48:
49:        outp(CMD_REG,com);
50:        for (i=0;i<max;i++)
51:            para(buff+i);
52:    }
53:
54:    para(data) /* FIFOにパラメータを送る */
55:    int data;
56:    {
57:        read:
58:        if(!inp(STS_REG)&0x02)
59:            outp(STS_REG,data);
60:        else
61:            goto read;
62:    }

```

メモリに連続して書き込めば、高速なグラフィックスの描画を行ってくれるように設計されています。

この場合、FIFOメモリのアドレスを読み込むことにより、FIFOメモリの状態(ステータス)が図2-1のように返されますので、FIFOに空きのある状態でパラメータに書き込みを行います。

表2-1、表2-2、表2-3がGDCのコマンドとパラメータの一覧表です。このコマンドやパラメータの詳細は誌面の関係で省略しますので、文献(1)などを参考にしてください。

リスト2-1はGDCアクセスのテスト・プログラムです。このプログラムでは、指定された範囲を指定された色(RGB)により塗りつぶします。また、このプログラムを少し変更すれば希望のパターンで塗りつぶすことも可能です。

このように、GDCやFDCなど高機能なコントローラを直接制御するには、ある程度の専門知識を必要と

しますので、先にも述べたように、一般的な応用にはBIOやLIOが利用されています。

## 2-2 グラフLIOの詳細

前述のように、ハードウェアを直接に制御し、より上位のプログラムからコールされる制御プログラムをBIOと呼んでいます。グラフィックの場合も**グラフBIO**があります。そして、この**グラフBIO**を利用した、より論理的なグラフィック処理ルーチンがROM内に収められており、この処理ルーチンを**グラフLIO**と呼んでいます。

グラフLIOには、表2-4のように**17種のコマンド**が用意されていて、通常、**割り込み番号A0h~AFh**および**CEh**の割り込みベクタを介してコールされます。また、この割り込みエントリはROM上の**F9900h番地から4バイトおきに格納**されています。したがって、このグラフLIOのユーザは、あらかじめ、



〈表2-4〉 グラフLIOの入出力パラメータ

割り込み番号	ルーチン名	機能	入力パラメータ	出力パラメータ
A0h	GINIT	初期化	なし	AH=00h:正常終了
A1h	GSCREEN	モード設定	BX:パラメータ・リストへのポインタ(図7)	AH=00h:正常終了, AH=05h:不正呼び出し
A2h	GVIEW	ビューポート指定	BX:パラメータ・リストへのポインタ(図8)	AH=00h:正常終了, AH=05h:不正呼び出し
A3h	GCOLOR1	背景色指定	BX:パラメータ・リストへのポインタ(図9)	AH=00h:正常終了
A4h	GCOLOR2	パレット番号と表示色の対応	BX:パラメータ・リストへのポインタ(図10)	AH=00h:正常終了
A5h	GCLS	描画領域の塗りつぶし	なし	AH=00h:正常終了
A6h	GPSET	点を打つ	BX:パラメータ・リストへのポインタ(図11) AH=01h:フォア・グラウンド・パレット番号 AH=02h:バック・グラウンド・パレット番号	AH=00h:正常終了
A7h	GLINE	直線, 方形を描く	BX:パラメータ・リストへのポインタ(図12)	AH=00h:正常終了
A8h	GIRCLE	円, 楕円を描く	BX:パラメータ・リストへのポインタ(図13)	AH=00h:正常終了, AH=06h:演算オーバフロー
A9h	GPAINT1	色で塗りつぶし	BX:パラメータ・リストへのポインタ(図14)	AH=00h:正常終了, AH=05h:不正呼び出し AH=07h:ワーク域不足
AAh	GPAINT2	タイルで塗りつぶし	BX:パラメータ・リストへのポインタ(図15)	AH=00h:正常終了, AH=05h:不正呼び出し AH=07h:ワーク域不足
ABh	GGET	描画情報の格納	BX:パラメータ・リストへのポインタ(図16)	AH=00h:正常終了, AH=05h:不正呼び出し
ACH	GPOT1	描画情報の表示	BX:パラメータ・リストへのポインタ(図17)	AH=00h:正常終了, AH=05h:不正呼び出し
ADh	GPOT2	日本語の描画	BX:パラメータ・リストへのポインタ(図18)	AH=00h:正常終了, AH=05h:不正呼び出し
AEh	GROLL	描画面面の移動	BX:パラメータ・リストへのポインタ(図19)	
AFh	GPOINT2	ドットのパレット番号の取得	BX:パラメータ・リストへのポインタ(図20)	AH=00h:正常終了, AL:パレット番号
CEh	GCOPY	ドット情報の格納	AX:左上点X座標 BX:左上点Y座標 CL:X方向ドット数 CH:Y方向ドット数 DI:バッファのオフセット・アドレス ES:バッファのセグメント・アドレス	AH:不定

この各コマンドの割り込みエントリを, 各割り込みベクタ・テーブルにセットしなければなりません。

ここでは, このグラフLIOの応用例としてLattice C(MS-DOS版V2.14, Sモデルに限定)により, グラフィック・ライブラリを作成します(INT CEhルーチンは除く)。

今回使用したMS-DOSシステムでは, 割り込み番号A0h~AFhは使用されていないので, グラフLIOの割り込みベクタをセットするだけになりましたが, **すでにA0h~AFhの割り込みベクタが使用されているような場合には, 一度ユーザのバッファにこれらのベクタ・テーブルを格納しておき, グラフィック処理が終わった時点で元のベクタ・テーブルに復元する必要があります。**

グラフLIOのユーザは, これらA0h~AFh, CEh以外にも注意しなければならない割り込み番号があります。グラフLIOでは比較的時間のかかる**コマンド実行中にも, ほかの割り込み処理が行えるように, 時々割り込み番号C5hのルーチンをコールしています。**

したがって, ユーザがこのC5h割り込みルーチンの中で, 例えばSTOPキーの監視やほかの割り込み処理を行うことなども可能になっています。

今回のグラフィック・ライブラリでは, このC5h

割り込み処理ルーチンでは何もしないことにし, IRET命令コード(CFh)をスタティック変数として定義し, その変数アドレスを, このC5h割り込みベクタとしてセットしています。

また, グラフLIOを使用する場合の準備として, 割り込みベクタ・テーブルのセットに加えて, **ワーク・エリアの確保**もしなければなりません。

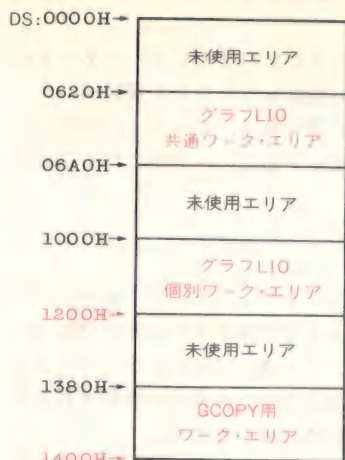
グラフLIOのワーク・エリアは図2-2のようにGCOPY(INT CEhルーチン)を使用しない場合に1200hバイト, GCOPYルーチンを使用する場合は, 1400hバイトを用意しなければなりません。

そして, このワーク・エリアは, **必ずデータ・セグメント(DSレジスタ)のオフセット0000h番地から配置**されなければなりません。このワーク・エリアのアドレスは, GINIT(A0hルーチン)コマンドをコールする際にグラフLIOに渡され, 以後の各コマンドをコールする際にも, DSレジスタはこのワーク・エリアを指していなければなりません。

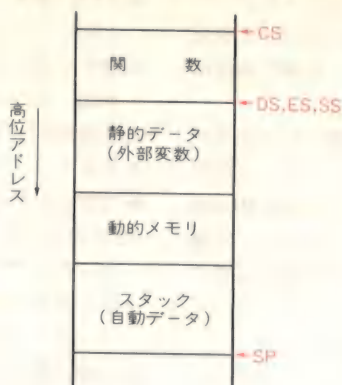
また, **パラメータ・リストは, このデータ・セグメント内に配置**されていなければならず, そのポインタとしては, BXレジスタが使われています。

これらのグラフLIOの各コマンドでは, DS, SS, SPの3個のレジスタは保証されていますが, その他のレジスタは保存されませんので注意が必要です。

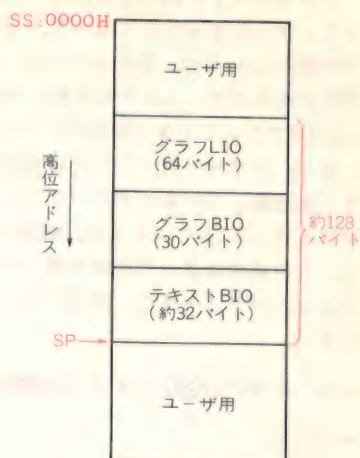
〈図2-2〉 グラフLIOのワーク・エリア



〈図2-3〉 Lattice Cにおけるメモリ領域とレジスタ



〈図2-4〉 グラフLIOのスタック・エリア



Lattice C(Sモデル)では、変数はデータ・セグメント(DS)内に収まっていて、このセグメントはプログラムの実行中は変化しませんが、関数の内部変数として、このグラフLIOのワーク・エリアを用意すると、データ・セグメントの最初からは配置されないのでは不都合が生じます(図2-3)。

したがって、これを解決するために、このグラフLIOのワーク・エリアを外部変数として定義すると、データ・セグメントの先頭から配置され、グラフLIOのワーク・エリアの仕様が満足されます(実際には、最初の約200バイトはmain関数により使用されますが、ここはグラフLIOの未使用エリアになっているので問

題ない)。

また、グラフLIOのスタック・エリアとして、図2-4のように約128バイト(図2-4はスタック内の概念的な使用状況であり、その内容は順不同になる)を用意しなければなりません。

これもLattice Cではデフォルトで、2048バイトのスタック・エリアが確保されるので、内部(自動)変数を大きく取らない限り問題ありません。また、内部(自動)変数に大きな配列を使うような場合には、\_stack変数に必要なバイト数を定義することにより自由に設定できます。詳しくは言語のマニュアルを参照してください。

## VシリーズのグラフLIOの活用法

Vシリーズ(VF, VM)からグラフィック拡張機能として、アナログRGB対応となっています。ハードウェアに関してはよくなったと思いますが、この機能を利用する方法としては、Vシリーズを購入すると必然的についてくるDISK-BASICを利用するしかありません。

また、Vシリーズになっても、下位機種との互換性を図るために、従来のデジタルRGB用のグラフLIOをそのままROM上(セグメント F990h ~)にもっています。

したがって、VシリーズではDISK-BASICを起動する最中にVシリーズかどうかを判断して、アナログRGB用のグラフLIOルーチン(約16Kバイト)をシステム・ディスクからメモリ上にロードしているようです。そこで、DISK-BASIC以外でも使えるように、この部分を少し拝借しようというわけです。

DISK-BASICおよびMS-DOS版BASIC上で、グ

ラフLIOをファイルに落とすプログラムをリストAに示します。

この部分を使用するにあたって気をつける点は、COLORに関してLIOをコールする際のパラメータの引き渡しです。障害が発生すると思われるのはCOLOR文だけで、一つはアナログRGBかどうかの切り替えを行うスイッチが追加されたことによるもの、もう一つはパレットを変更するときの値です。パレットに関しては、受け渡しの際の2番目の引き数が従来1バイトであったのが2バイトになったためです。

〈沢井孝裕〉

〈リストA〉 グラフLIOをファイルに落とすプログラム

```

1000 '
1010 ' 注) 本プログラムを実行する環境としては、
1020 '
1030 '      N88-BASIC(86) [DISK-BASIC版]
1040 '      " " [MS-DOS版]
1050 '
1060 '      のどちらでも構いませんが、バージョンは3.0に限ります。
1070 '
1080 DEF SEG=0
1090 LIOSEG = PEEK(&H282)+PEEK(&H283)*256
1100 DEF SEG=LIOSEG
1110 BSAVE "ARGLIO",0,&H3FFF
1120 PRINT "Complete !!"
1130 END
    
```



これらのグラフLIOの各コマンドで使用するパレット・コードやカラー・コード、その他のパラメータや用語については、BASICコマンドのそれとほぼ互換性があるので、ここでは省略します。必要な場合はBASICマニュアルなどを参照してください。

また、カラー・コードなどは対象機種としてE,Fタイプを想定しているので、パラメータ・リストの説明では8色になっていますが、機種によって16/4096色モードの場合は多少の変更を要します。そして、座標点などのパラメータは、矛盾のないように指定してください。

## 2-3 グラフLIOのコマンドの詳細

### ▶ 初期化

(GINIT:A0h)

このルーチンでは、グラフLIO(ワーク・エリアやGDCなど)の初期化が行われます。グラフLIOを使用する場合は、最初に必ずこのコマンドを実行しなければなりません。リスト2-2では、このGINIT関数内でベクタ・テーブルの初期化も行っています。

### ▶ モード設定

(GSCREEN:A1h)

グラフィック画面のモード設定を行います(図2-5)。

### ▶ 描画領域の指定

(GVIEW:A2h)

〈図2-5〉 GSCREENのパラメータ・フォーマット

	BX+0	+1	+2	+3	+4
	画面 モード	画面 スイッチ	アクティブ・ ページ	ディスプレイ・ ページ	
パラ メータ	画面モード	画面スイッチ	アクティブ・ ページ	ディスプレイ・ ページ	
00h	カラー・モード (640×200ドット)	表示あり	アクティブ・ ページの番号 (0～11)	ディスプレイ・ ページの番号 (0～31)	
01h	モノクロ・モード (640×200ドット)	表示あり 高速書き込み			
02h	モノクロ・モード (640×400ドット)	表示なし			
03h	カラー・モード (640×400ドット)	表示なし 高速書き込み			
FFh	今までのモードを引き継ぐ				

〈図2-6〉 GVIEWのパラメータ・フォーマット

BX+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
左上X座標 (X <sub>1</sub> )	左上Y座標 (Y <sub>1</sub> )	右下X座標 (X <sub>2</sub> )	右下Y座標 (Y <sub>2</sub> )	領域 色	境界 色					
					パレット番号 (0~7) 外枠を描かない(F'F'h)					
					パレット番号(0~7) 塗りつぶししない(F'F'h)					

アクティブ画面内の描画領域(ビュー・ポート)の指定を行います。

このコマンドでは、必要に応じてビュー・ポート内の塗りつぶしや外枠の描画も行われます(図2-6)。

また、このコマンド実行後は、アクティブ画面への図形描画はビュー・ポート内だけにのみ反映されることになります。

### ▶ 背景色などの指定

(GCOLOR1:A3h)

バック・グラウンド・カラー、ボーダ・カラー、フォア・グラウンド・カラーの指定を行います(図2-7)。

バック・グラウンド・カラーとは、グラフィック画面の地の色で、この後のGCLS命令が実行されたときにここで指定された色に変わります。

ボーダ・カラーは640×400ドット・モードの場合は意味がありません。

フォア・グラウンド・カラーは、パレット番号省略時に使用される色です。

また、Vシリーズでは、図2-7のパラメータの後にさらに1バイトのパレット・モードのコードが追加されます。

### ▶ パレット番号と表示色の対応

(GCOLOR2:A4h)

パレット番号を、指定された表示色コードに対応させます(図2-8)。

〈図2-7〉 GCOLOR1のパラメータ・フォーマット

BX+0	+1	+2	+3	+4
未使用	バック・ グラウンド・ カラー (0~7)	ボーダ・ カラー (0~7)	フォア・ グラウンド・ カラー (0~7)	

〈図2-8〉 GCOLOR2のパラメータ・フォーマット

BX+0	+2	+3
パレット 番号 (0~7)	表示色 コード (0~7)	

これにより、すでに描画済みの表示色も変更した表示色になります。また、Vシリーズの16/4096モードでは表示色コードは3バイトが必要になります。

#### ▶描画領域の塗りつぶし

(GCLS:A5h)

アクティブ画面内の描画領域をバック・グラウンド・カラーで塗りつぶします。このコマンドにはパラメータはありません。

#### ▶点の描画

(GPSET:A6h)

指定された座標に指定された色でドットを描きます(図2-9)。

#### ▶直線/方形の描画

(GLINE:A6h)

指定された2点を結ぶ直線、あるいはこれを対角線

〈図2-9〉 GPSETのパラメータ・フォーマット

BX +0	+1	+2	+3	+4	+5
X座標		Y座標		パレット番号 (0~7)	

〈図2-10〉 GLINEのパラメータ・フォーマット

BX +0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14
始点の X座標(X <sub>1</sub> )		始点の Y座標(Y <sub>1</sub> )		終点の X座標(X <sub>2</sub> )		終点の Y座標(Y <sub>2</sub> )		パレット番号1 (0~7)	描画指定 0:直線 1:方形 塗りつぶし	ライン・スタイル・スイッチ	パレット番号2	ライン・スタイル LOW	パターンの長	ハイ

#### ▶ライン・スタイル・スイッチ(+10)

- 00h: 指定なし
- 01h: ライン・スタイルまたはパレット番号2指定あり
- 02h: タイル・パターン指定あり

#### ▶パレット番号2(+11)

- 方形内部の塗りつぶし色指定
- 描画コード=20hのみ有効

#### ▶ライン・スタイル(+11, +12)

b <sub>0</sub>	b <sub>7</sub>	b <sub>8</sub>	b <sub>15</sub>
LO(+11)		HI(+12)	

タイル・パターン格納域	
オフセット・アドレス	セグメント・アドレス

〈図2-11〉 GCIRCLEのパラメータ・フォーマット

BX	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14											
	中心点 X 座標(CX)		中心点 Y 座標(CY)		X 方向半径 (RX)		Y 方向半径 (RY)		パレット 番号1 (0~7)	フラグ		開始点 X 座標(SX)		開始点 Y 座標(SY)												
															+14	+15	+16	+17	+18	+19	+20	+21	+22	+23		
															終了点 X 座標(EX)		終了点 Y 座標(EY)		パレット番 号2 or タイル・ パターン長	タイル・パターン格納域 オフセット・アドレス・セグメント・アドレス						

#### ▶フラグ(+9)

- b<sub>0</sub>: 開始点指示(0: なし 1: あり)
- b<sub>1</sub>: 開始線分指示(0: なし 1: あり)
- b<sub>2</sub>: 終了点指示(0: なし 1: あり)
- b<sub>3</sub>: 終了線分指示(0: なし 1: あり)

- b<sub>4</sub>: 描画方法(0: 全楕円を描画, 1: 1点のみ描画)
- b<sub>5</sub>: 塗りつぶし指示(0: なし 1: あり)
- b<sub>6</sub>: タイル・パターン指示(0: なし 1: あり)

とする方形の描画を行います。

また、必要によりライン・スタイルや方形内の塗りつぶし(色やタイル・パターン)も行うことができます(図2-10)。

ライン・スタイルやタイリング・データの詳しいことについてはBASICマニュアルを参照してください。

#### ▶円/楕円の描画

(GCIRCLE:A7h)

指定された中心点、半径(X方向、Y方向)の円や楕円の描画を行います。

また、開始点、終了点の指定により円弧や扇型も描画でき、これらの円や扇型の内部を指定した色、またはタイルで塗りつぶすこともできます(図2-11)。

#### ▶ペインティング

(GPAINT1:A9h)

指定した点と境界色で決定される領域を、指定した色で塗りつぶします(図2-12)。

図2-12でのワーク領域の必要な大きさはペインティング領域の大きさにより変化するので、ある程度大きめに確保します。



〈図2-12〉 GPAINT1のパラメータ・フォーマット

BX +0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
開始点 X座標(X)		開始点 Y座標(Y)		領域色 パレット 番号	境界色 パレット 番号	ワーク域 最終オフセット		先頭オフセット		

- ▶ ワーク域は16バイト以上
- ▶ ワーク域はデータ・セグメント内にあること

〈図2-13〉 GPAINT2のパラメータ・フォーマット

BX +0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+16	+17	+18	+19	+20
開始点 X座標(X)		開始点 Y座標(Y)		未使用	タイル・ パターン 長さ (バイト)	オフセット・アドレス	タイル・パターン格納域 セグメント・アドレス		境界色 パレット 番号 (0~7)	未使用	ワーク域 最終オフセット		先頭オフセット			

図14参照

〈図2-14〉 GGETのパラメータ・フォーマット

BX +0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14
左上点 X座標(X1)		左上点 Y座標(Y1)		右下点 X座標(X2)		右下点 Y座標(Y2)		格納域 オフセット・アドレス	格納域 セグメント・アドレス		格納域 長さ			

〈図2-15〉 GPUT1のパラメータ・フォーマット

BX +0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14
左上点 X座標(X)		左上点 Y座標(Y)		格納域 オフセット・アドレス	格納域 セグメント・アドレス		格納域 長さ(バイト)	描画 モード	カラー・ スイッチ	フォア・ グラウンド・ カラー	バック・ グラウンド・ カラー (0~7)			

▶ 描画モード(+10)

指定領域上のパターンと  
格納域パターンとの論理演算  
00h: T  
01h: NOT  
02h: OR  
03h: AND  
04h: XOR

〈図2-16〉

GPUT2のパラメータ・フォーマット

BX +0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
左上 X座標(X)		左上 Y座標(Y)		日本語JIS コード	描画 モード	カラー・ スイッチ	フォア・ グラウンド・ カラー	バック・ グラウンド・ カラー (0~7)		

▶ 描画モード(+6)

指定領域上のパターンと  
格納域パターンとの論理演算  
00h: T  
01h: NOT  
02h: OR  
03h: AND  
04h: XOR

▶ タイリング

(GPAINT2: AAh)

指定した点と境界色で決定される領域を指定された  
タイル・パターンで塗りつぶします(図2-13)。

▶ 描画情報の格納

(GGET: ABh)

指定領域の描画情報を指定されたバッファに格納し  
ます(図2-14)。

▶ 格納された描画情報の表示

(GPUT1: Ach)

格納されている描画情報を指定された領域に描画し  
ます(図2-15)。

▶ 日本語の描画

(GPUT2: ADh)

JISコードで指定された日本語を描画します(図2-  
16)。

▶ 描画面面の移動

(GROLL: AEh)

アクティブ画面全体の描画情報を指定されたドット  
数だけ上下左右方向にスクロールします(図2-17)。

## 〈リスト2-3〉 グラフLIOのテスト・プログラム

```

1:
2: /* グラフィック-LIO テストプログラム
3:    1988-8 H.ABE */
4:
5: #define COL_MAX 8
6:
7: char buff_luo[0x1400]; /* G-LIO ワークエリア */
8:
9: main()
10: {
11:     puts( "Y033#Y033= " ); /* テキスト画面の消去 */
12:     ginit();
13:     gscreen(3,0,0,1); /* 640*400 ドット */
14:     gcls();
15:     box(); /* 箱 */
16:     scircle(); /* 楕円 */
17:     roll_l(); /* 画面の左移動 */
18:     ginit();
19:     gscreen(3,0,0,1);
20: }
21:
22: box() /* 箱のぬりつぶし */
23: {
24:     int i,j,y1=0,y2=0;
25:
26:     for (i=1;i<COL_MAX;i++){
27:         y2=y1+50;
28:         gline(400,y1,600,y2,i,0); /* 箱を描く */
29:         y1=y1+55;
30:     }
31:     for (j=1;j<COL_MAX;j++){
32:         y1=0;
33:         for (i=1;i<COL_MAX;i++){
34:             itaval(2);
35:             gpaintl(410,y1+10,i,j,i); /* 箱を塗りつぶす */
36:             y1=y1+55;
37:         }
38:     }
39: }
40:
41: scircle() /* 楕円を描く */
42: {
43:     int i,j,col2;
44:     for(i=1;i<COL_MAX;i++){
45:         gcircle(200,200,200-i*20,100-i*10,i,0x20,i+1);
46:         itaval(10);
47:     }
48:     for(j=1;j<400;j++){
49:         for(i=1;i<COL_MAX;i++){
50:             col2=rand()*X7+1;
51:             gcolor2(i,col2); /* パレット番号を指定 */
52:             gcolor2(col2,i); /* 元の色に */
53:         }
54:     }
55: }
56: roll_l() /* 画面を左に移動する */
57: {
58:     int i;
59:
60:     for(i=0;i<17;i++){
61:         groll(0,40,0);
62:     }
63: }
64: itaval(count) /* インターバル */
65: int count;
66: {
67:     int i,j;
68:     for (j=1;j<count+1;j++){
69:         for (i=0;i<10000;i++){
70:             ;
71:         }
72:     }
73: }

```

〈図2-17〉 GROLLのパラメータ・フォーマット

BX+0    +1    +2    +3    +4    +5

上下ドット数 (-399~399)	左右ドット数 (-639~639)	フラグ
----------------------	----------------------	-----

### ▶フラグ(+4)

- 00h: 移動後の残り領域を  
パレット0にする
- 01h: 移動後の残り領域を  
バック・グラウンド・  
カラーにする

〈図2-18〉  
GPOINT2の  
パラメータ・  
フォーマット

BX+0    +1    +2    +3    +4

X座標,(X)	Y座標,(Y)
---------	---------

### ▶ドットのパレット番号の通知

(GPOINT2: A F h)

指定座標のドットのパレット番号をALレジスタに返します(図2-18)。

## 2-4 GDCのテスト・プログラム

これらのグラフィック・ライブラリの使用例をリスト2-3に示します。

このプログラムでは、グラフィック画面の初期化を行った後、7個の箱を描いてその色をペインティングにより変化させます。リスト2-3では、この際の処理が早すぎて見栄えがしないので、少しインターバルを取りながらペインティングを繰り返しています。

次に7個の同心楕円を色を変えながら描き、描き終わった時点で、パレット番号を書き換えることにより、あたかも画面がフラッシュしているかのごとく表現しています。そして、フィナーレでは画面を左にスクロールして終了します。

これにBASICのLOCATE文にあたる関数を、エスケープ・シーケンスを利用して作成すれば、BASIC感覚のグラフィックスが高速に表現できますので、処理結果を視覚的に把握したいようなアプリケーションには重宝するものと思います。

また、今回は比較的簡単にテストするため、Sモデルに限定したライブラリの作成を行いました。ポイント関連のアクセス法を変更すれば、Dモデルなどにも十分に対応できますので、挑戦してみてください。

ここでは、パラメータのチェック(カラー・コードなど)は行っていないが、グラフLIOの不当なアクセスを防ぐため、これらのパラメータのチェック・ルーチンを組み込んだほうが、より実用的なプログラムになるでしょう。

### ●参考文献●

- (1) 日本電気、μPD7220GDCユーザーズ・マニュアル。
- (2) B.W.カーニハン他；プログラミング言語C、共立出版。
- (3) 中村和朗訳；Lattice Cの使い方、工学図書。
- (4) 阿部英志；MS-DOS活用テクノロジー、マイコンピュータNo20、CQ出版社。
- (5) C on the PC98、ソフトマインド(1)、CQ出版社。



```

1: #include <dos.h>
2:
3: /* G - L I O コマンドライブラリ
4:      1986-8 H.ABE */
5:
6: #define LIO_SEG 0xf90
7: #define WORK_MAX 100
8:
9: ginit() /* グラフLIOの初期化 */
10: {
11:     init(); /* ベクタの初期化 */
12:     sys_call(0xa0);
13: }
14:
15: gscreen(mode,sw,act,dsp) /* グラフィック画面のモード設定 */
16: {
17:     int mode,sw,act,dsp; /* 画面モード,スイッチ,7セグ画面,2セグ画面 */
18:     char buff[4];
19:
20:     buff[2]=act;
21:     buff[1]=sw;
22:     buff[0]=mode;
23:     buff[3]=dsp;
24:     sys_call(0xa1,buff);
25: }
26:
27: gview(x1,y1,x2,y2,col1,col2) /* 描画領域の指定 */
28: {
29:     int x1,y1,x2,y2,col1,col2; /* 左上x1,y1,右下x2,y2,領域色,境界色 */
30:     int buff[4];
31:     char buff1[2];
32:
33:     buff[0]=x1;
34:     buff[1]=y1;
35:     buff[2]=x2;
36:     buff[3]=y2;
37:     buff1[0]=col1;
38:     buff1[1]=col2;
39:     sys_call(0xa2,buff);
40: }
41:
42: gcolor1(col1,col2,col3) /* 背景色等の指定 */
43: {
44:     int col1,col2,col3; /* バックグラウンド,ボーダー,フォアグラウンド */
45:     char buff[4];
46:
47:     buff[1]=col1;
48:     buff[2]=col2;
49:     buff[3]=col3;
50:     sys_call(0xa3,buff);
51: }
52:
53: gcolor2(pal,col) /* パレット番号と表示色コードの対応 */
54: {
55:     int pal,col; /* パレット番号,表示色 */
56:     char buff[2];
57:
58:     buff[0]=pal;
59:     buff[1]=col;
60:     sys_call(0xa4,buff);
61: }
62:
63: gcls() /* 描画領域の塗りつぶし */
64: {
65:     sys_call(0xa5);
66: }
67:
68: gposet(x,y,col,mode) /* 点を打つ */
69: {
70:     int x,y,col,mode; /* 座標(x,y),表示色,モード */
71:     int buff[2];
72:     char buff1[1];
73:
74:     buff[0]=x;
75:     buff[1]=y;
76:     buff1[0]=col;
77:     sys_call(0xa6,buff,mode);
78: }
79:
80: gline(x1,y1,x2,y2,pal1,code,sw,p1,p2,p3) /* 直線/矩形を描く */
81: {
82:     int x1,y1,x2,y2,pal1,code,sw; /* 始点(x1,y1),終点(x2,y2),境界色,スイッチ */
83:     unsigned int p1,p2,p3; /* p1=領域色 or 4セグ長 or 9セグ長
84:                               p2=格納域アドレス, p3=格納域のセグメント */
85:     int buff[4];
86:     char buff1[6];
87:     int buff2[2];
88:
89:     switch(sw){
90:     case 0:
91:         break;
92:     case 1:
93:         if(code==2)
94:             buff1[3]=p1;
95:         else
96:             buff1[3]=p1*0x100;
97:             buff1[4]=p1*0x100;
98:         break;
99:     case 2:
100:         buff1[5]=p1;
101:         buff2[0]=p2;
102:         buff2[1]=p3;
103:         break;
104:     }
105:     buff[0]=x1;
106:     buff[1]=y1;
107:     buff[2]=x2;
108:     buff[3]=y2;
109:     buff1[0]=pal1;
110:     buff1[1]=code;
111:     buff1[2]=sw;
112:     sys_call(0xa7,buff);
113: }
114:
115: gcircle(cx,cy,rx,ry,pal,flg,p1,p2,p3,p4,p5,p6,p7) /* 円,楕円を描く */
116: {
117:     int cx,cy,rx,ry,pal,flg; /* 中心(cx,cy),半径(rx,ry),境界色,フラグ */
118:     unsigned int p1,p2,p3,p4,p5,p6,p7; /* 領域色(p1)
119:                                           or 9セグ長(p1),格納域アドレス(p2),セグメント(p3)
120:                                           or 開始点(p1,p2),終了点(p3,p4)
121:                                           or 開始点(p1,p2),終了点(p3,p4),領域色(p5)
122:                                           or 開始点(p1,p2),終了点(p3,p4),9セグ長(p5)
123:                                           or 格納域オフセット(p6),セグメント(p7) */
124:
125:     {
126:         int buff[4];
127:         char buff1[2];
128:         int buff2[4];
129:         char buff3[3];
130:         int buff4[1];
131:
132:         if(flgs & 0x03){
133:             buff2[0]=p1; /* 開始,終了あり */
134:             buff2[1]=p2;
135:             buff2[2]=p3;
136:             buff2[3]=p4;
137:             if(flgs & 0x40){
138:                 buff3[0]=p5; /* タイル */
139:                 buff3[1]=p6*0x100;
140:                 buff3[2]=p6*0x100;
141:                 buff4[0]=p7;
142:             }
143:             else if(flgs & 0x20)
144:                 buff3[0]=p5; /* ベイント */
145:                 /* なし */
146:             }
147:             else if(flgs & 0x40){
148:                 buff3[0]=p1; /* 開始,終了なし */
149:                 buff3[1]=p2*0x100;
150:                 buff3[2]=p2*0x100;
151:                 buff4[0]=p3;
152:             }
153:             else if(flgs & 0x20)
154:                 buff3[0]=p1; /* ベイント */
155:                 buff[0]=cx;
156:                 buff[1]=cy;
157:                 buff[2]=rx;
158:                 buff[3]=ry;
159:                 buff1[0]=pal;
160:                 buff1[1]=flg;
161:                 sys_call(0xa8,buff);
162:             }
163:
164:             gpaint1(x,y,pal1,pal2) /* 塗りつぶし色で行う */
165:             {
166:                 int x,y,pal1,pal2; /* 領域(x,y),領域色,境界色 */
167:                 int buff[2];
168:                 char buff1[2];

```

リスト2-2) グラフィック・ライブラリ(つづき)

```

169: int buff2[2];
170: char work[WORK_MAX];
171:
172: buff[0]=x;
173: buff[1]=y;
174: buff[0]=pal1;
175: buff[1]=pal2;
176: buff2[0]=(int)&work+WORK_MAX;
177: buff2[1]=(int)&work;
178: sys_call(0xaa,buff);
179: }
180:
181: gpaint2(x,y,len,b_off,b_seg,pal) /* タイルパターンでの塗りつぶし */
182: int x,y,len,pal; /* 開始(x,y),h*タン長,境界色 */
183: unsigned int b_off,b_seg; /* 格納アドレス,セグメント */
184: {
185:     int buff[2];
186:     char buff1[2];
187:     int buff2[2];
188:     char buff3[6];
189:     int buff4[2];
190:     char work[WORK_MAX];
191:
192:     buff[0]=x;
193:     buff[1]=y;
194:     buff[1]=len;
195:     buff2[0]=b_off;
196:     buff2[1]=b_seg;
197:     buff3[0]=pal;
198:     buff4[0]=(int)&work+WORK_MAX;
199:     buff4[1]=(int)&work;
200:     sys_call(0xaa,buff);
201: }
202:
203: gset(x1,y1,x2,y2,b_off,b_seg,len) /* 描画情報の格納 */
204: int x1,y1,x2,y2; /* 左上(x1,y1),右下(x2,y2) */
205: unsigned int b_off,b_seg,len; /* h*タン長,セグメント,長さ */
206: {
207:     int buff[7];
208:
209:     buff[0]=x1;
210:     buff[1]=y1;
211:     buff[2]=x2;
212:     buff[3]=y2;
213:     buff[4]=b_off;
214:     buff[5]=b_seg;
215:     buff[6]=len;
216:     sys_call(0xab,buff);
217: }
218:
219: gputl(x,y,b_off,b_seg,len,mode,sw,col1,col2) /* 描画情報の表示 */
220: int x,y; /* 左上(x1,y1),右下(x2,y2) */
221: int mode,sw,col1,col2; /* モード,サイズ,スタイル,色 */
222: unsigned int b_off,b_seg,len; /* h*タン長,セグメント,長さ */
223: {
224:     int buff[5];
225:     char buff1[4];
226:
227:     if(sw==1){
228:         buff1[2]=col1;
229:         buff1[3]=col2;
230:     }
231:     buff[0]=x;
232:     buff[1]=y;
233:     buff[2]=b_off;
234:     buff[3]=b_seg;
235:     buff[4]=len;
236:     buff1[0]=mode;
237:     buff1[1]=sw;
238:     sys_call(0xac,buff);
239: }
240:
241: gput2(x,y,code,mode,sw,col1,col2) /* 日本語の表示 */
242: int x,y; /* 左上(x1,y1) */
243: int mode,sw,col1,col2; /* モード,サイズ,スタイル,色 */
244: unsigned int code; /* jisコード */
245: {
246:     int buff[3];
247:     char buff1[4];
248:
249:     if(sw==1){
250:         buff1[2]=col1;
251:         buff1[3]=col2;
252:     }
253:     buff[0]=x;
254:     buff[1]=y;
255:     buff[2]=code;
256:     buff1[0]=mode;
257:     buff1[1]=sw;
258:     sys_call(0xad,buff);
259: }
260:
261: groll(v,h,flag) /* 描画面面の移動 */
262: int v,h,flag; /* 縦(v),横(h),方向 */
263: {
264:     int buff[2];
265:     char buff1;
266:
267:     buff[0]=v;
268:     buff[1]=h;
269:     buff1=flag;
270:     sys_call(0xae,buff);
271: }
272:
273: gpoint2(x,y) /* ポイントに対応するh*タン番号の通知 */
274: int x,y; /* ポイントの座標(x,y) */
275: {
276:     int buff[2];
277:
278:     buff[0]=x;
279:     buff[1]=y;
280:     return(sys_call(0xaf,buff));
281: }
282:
283: init() /* ベクタのセット */
284: {
285:     static char iret_codes=0xcf;
286:     struct SREGS segregs;
287:     int i,buff;
288:
289:     for (i=0; i<64; i=i+4){
290:         peek(LI0_SEG,i+6,&buff,2);
291:         poke(0,i+0xa0+4,&buff,2);
292:         buff=LI0_SEG;
293:         poke(0,i+0xa0+4+2,&buff,2);
294:     }
295:     poke(0,0xc0+4+2,&buff,2);
296:     peek(LI0_SEG,70,&buff,2);
297:     poke(0,0xc0+4,&buff,2);
298:     segread(&segregs);
299:     buff=(int)&iret_code;
300:     poke(0,0xc5+4,&buff,2);
301:     buff=segregs.ds;
302:     poke(0,0xc5+4+2,&buff,2);
303: }
304:
305: sys_call(vect.cmd,ah) /* G-LI0 システムコール */
306: int vect; /* ベクタ番号 */
307: char ah,cmd; /* AHレジスタ,パラメータポインタ */
308: {
309:     union REGS regset;
310:     struct SREGS segregs;
311:
312:     segread(&segregs);
313:     regset.x.bx=(int)cmd;
314:     regset.h.ah=ah;
315:     int8x(vect,&regset,&regset,&segregs);
316:     if(regset.h.ah !=0)
317:         error(vect,regset.h.ah);
318:     return(regset.x.ax);
319: }
320:
321: error(vect,ah) /* エラールーチン */
322: int vect; /* ベクタ番号 */
323: char ah; /* エラーステータス */
324: {
325:     printf("\nG-LI0 のアクセスエラーです。 %d vector no=%d\n",vect);
326:     printf(" error code=%d\n",ah);
327:     exit(1);
328: }

```



# MS-DOS上でのアセンブラ・プログラミング

本章では、MS-DOS上でのアセンブラによるプログラミングの方法およびMS-FORTRANとのリンクの方法について解説します。また、応用例としてアセンブラによるグラフィック・ライブラリを紹介します。

## 1 PC9801のオペレーティング・システム

### 1-1 OSの標準機能

16ビット・パーソナル・コンピュータではOS (Operating System) が標準的にサポートされます。現在、8086系CPUを対象として入手可能なOSはMS-DOS、CP/M86、RMX/86、XENIX、PC-UXそしてBASICインタプリタといったものです。

最近では、BASICは汎用OS上で提供されるものが多くなりましたが、PC9801用のN<sub>86</sub>BASICは現在でもスタンド・アロンで最低限のOSの機能を含んでいます。

BASICは別として、いわゆる汎用OSの機能は、

- ①システム管理プログラム(モニタ)
- ②ユーティリティ・プログラム群

に大別されます。図1-1はこれを概念的に図示したものです。最近では、さらにネットワーク機能を含むものもあります。これらの機能を備えたものがOSと呼ばれるわけです。また、①の機能のみに重点を置いて

設計されたものをモニタと呼ぶことがあります。

最近のように、ユーティリティ・プログラムが充実してきて、考えられるコマンドがすべてそろっているという状況になってくると(例えばUNIX)、それら全体を環境(Environment)と呼ぶこともあります。

OSが必要な理由は次のようなものです。

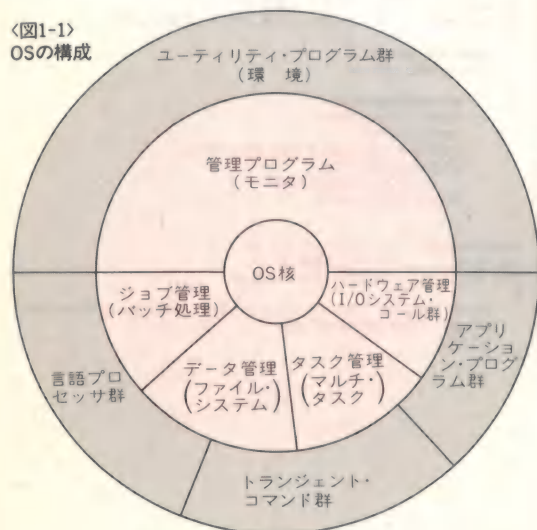
- ①16ビット・パーソナル・コンピュータの豊富なハードウェア資源を有効に利用する。
- ②物理的な入出力装置(ディスク装置、プリンタ、キーボードなど)を論理的なモデルに変換することによって、ソフトウェアからのインターフェースを容易にする。
- ③コンピュータと人間のやりとりの窓口となる(CCP, SHELL)。
- ④プログラム開発のツールが統一的に用意されるので、開発環境として効率が高い。
- ⑤共通のファイル・システムによって異なる機種間のデータ/プログラムの交換を可能にする。

論理の入出力機器のモデル化はいくつかの段階があります。基本的なインターフェースは、図1-2に示したようなシステム・コールの形で与えられます。OSがサポートする入出力機器については、物理的なタイミングやデータ単位を考えなくてもよいようなシステム・コールが用意されています。

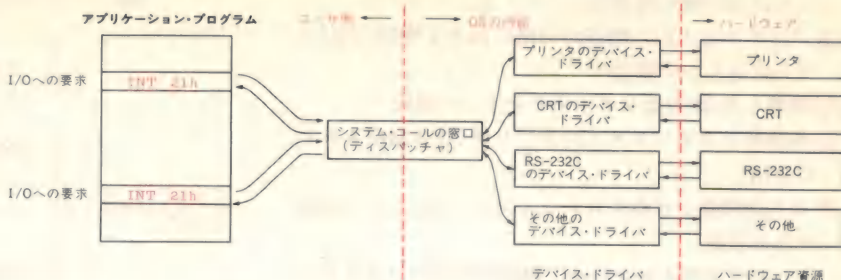
最近の傾向として、MS-DOSやUNIXではさらに抽象的なインターフェースとして、統一されたファイル・イメージのインターフェース(デバイス・ファイル)が与えられています。この場合は、機器に対応するファイル名を指定するだけで、ディスク上のファイルと同格にデータを交換できるようになり、ハードウェアの実体は何であるかをほとんど意識する必要がありません。

また、ある時点に何をしたいかをパーソナル・コンピュータに伝える手段、つまりコンピュータとのコミュニケーションの窓口が必要です。もちろん専用機ならばパワー-ON即スタートというスタイルも考えられますが、汎用計算機としてのパーソナル・コンピュータならばいったんコンソールからの入力待ち、それ

〈図1-1〉  
OSの構成



〈図1-2〉  
システム・コールによる  
入出力機器の操作



〈表1-1〉 8086/88用の代表的なOS

OS 名称	ユーザ数	同時実行 プロセス	特 徴
CP/M 86	1	1	バッチ指向
MS-DOS (V3.1)	1	1	UNIX スタイル
MS-Windows	1	1	マルチ・ウィンドウ
CCP/M	1	4	マルチタスク, マルチ・ウィンドウ
XENIX	複数	無制限	マルチ・ユーザ/マルチタスク
RMX/86	1	無制限	総合開発環境, リアルタイム指向

を解析して指定されたプログラムを起動します。

この機能をつかさどる部分は**CCP**とか**SHELL**と呼ばれます。このうち、SHELLという用語は制御言語に近い高度な機能をもった場合に使われます。

従来はコマンド・プロセッサはOSの一部として核に組み込まれていたのですが、UNIX以後のOSでは、**コマンド・プロセッサも普通のプログラムと同じように、OSの核が起動するプログラムとなっているもの**が多くなっています。CP/M86ではCCPとして組み込まれていますが、MS-DOSではSHELLに近いイメージの**コマンド・インタプリタ (COMMAND, COM)**としてこの機能が提供されます。

ファイル・システムの管理は、パーソナル・コンピュータ用のOSの最も重要な機能です。つまり、フロッピー・ディスクなどの可搬型のメディアであっても、記録形式(フォーマット)や記述形式(ファイル編成方式)が異なれば、データの交換をすることは容易ではありません。OSの仕様には、ファイルの記録形式が含まれますから、それに従ってさえいけば、異なるハードウェアでもデータを読み書きできます。

一般に**ファイルの記録形式は、同一のOS間で互換性があります。**

## 1-2 OSの分類

OSならばどれも同じというわけではありません。OSはその機能によってさらにいくつかのクラスに分類されます。

まず、同時に(見掛け上)実行されるタスクの数が1個かそれ以上かによって、**シングル・タスク/マルチタスク**かの分類がなされ、さらに同時に使用できるユーザの数が1人かそれ以上かによって、**シングル・ユーザ/マルチ・ユーザ**の分類がなされます。また、マルチ

タスクのOSにおいては、非同期的なタスクの切り替え要求にどう対応するかによって、**リアルタイム/非リアルタイム**の分類がされることもあります。

代表的OSについてこの分類をしたものが表1-1です。現在、16ビット・パーソナル・コンピュータに搭載されているOSは、まだ**シングル・ユーザ/シングル・タスク**のものがほとんどです。

## 1-3 MS-DOSの機能と構造

現在、8086/88を用いたパーソナル・コンピュータに最も広く採用されているOSはMS-DOSです。最も新しいバージョン(MS-DOS V3.1)では、表面的にはUNIXに極めて近いユーザ・インターフェースを与えています。しかし、機能的な分類では**シングル・ユーザ/シングル・プロセスの最も単純なOSです。割り込み処理などもOSの範囲外となっており、リアルタイム処理の管理は行えません。**

今後の改版では、マルチプロセスが取り入れられることになると思われます。また、すでにアナウンスされているMS-Windowsは、MS-DOSの拡張としてマルチ・ウィンドウの機能を取り入れたもので、基本的なシステム・コール、ファイル・システムについてはMS-DOSと互換性があります。

## 1-4 MS-DOS (V3.1) とUNIX

MS-DOS (V3.1) の特徴は、UNIXに近いユーザ・インターフェースがパーソナル・コンピュータでも得られることといえるでしょう。これは、開発元であるマイクロソフト社が上位OSとしてXENIX (UNIX) をリリースしている関係から、スタイルの統一をとるためともいえます。MS-DOSでは以下のような特徴を備えています。



- ① UNIXと同様の階層的なファイル・システム
- ② ファイルと入出力機器が同格に扱える機能(デバイス・ファイルの概念)
- ③ 標準入力/出力とリダイレクションの機能
- ④ 高性能なコマンド・インタプリタ(COMMAND.COM)
- ⑤ 日本語機能などのフロントエンド・プロセッサが組み込み可能

さらに付け加えるならば、現在8086上のソフトウェアは大半がMS-DOS上で開発されるため、その上で使用できるアプリケーション・ソフトウェアの数が抜群に多いことも特徴の一つといえるでしょう。

MS-DOSはシングル・プロセスのOSなので、平行プロセスとしてのサブプロセスを起動することはできませんが、いったん自分自身を中断する形で実行されるサブプロセスを起動することができます。

## 1-5 MS-DOSの構造とインターフェース

MS-DOSのメモリ・レイアウトを図1-3に示します。常駐する核の部分はファイル・バッファの量によって異なりますが、PC9801では約63Kバイトほどになっています。核の部分は、ハードウェアに依存するIO.SYS、ハードウェアに依存しないMSDOS.SYS、さらにCOMMAND.COMの常駐部に分かります。IO.SYSはCP/MのBIOSに相当する部分で、MS-DOSのインプリメントは、各ハードウェアに合わせてこの部分をカスタマイズすることになります。

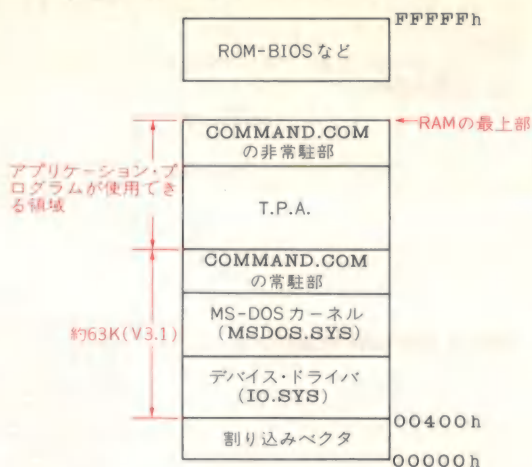
MSDOS.SYSはCP/MでのBDOSに相当し、OSの基本的な機能がすべてここに含まれています。COMMAND.COMはCP/MのCCPに相当する部分で、コンソールとの対話やバッチ・ジョブの管理を行います。MS-DOSではCOMMAND.COMは常駐部と非常駐部に分かれています。非常駐部はタスクの実行にメモリが不足するといった追い出され、有効メモリ領域を広げます。

つまり、COMMAND.COMの仕事のうち、コンソールとの対話の部分はいったん制御がアプリケーション・プログラムに移った以後は必要ないので、その分メモリを占有するのはもったいないからです。CP/M80でもCCPは同じように追いだされていました。

MS-DOSとアプリケーション・プログラムのインターフェースは、システム・コールの窓口を通して行われます。MS-DOSのシステム・コールはソフトウェア割り込み(INT 21hなど)を用いています。

MS-DOSではINT 20h~3Fhがシステム・コール用として予約されています。MS-DOSを搭載するパーソナル・コンピュータは、この領域をハードウェア割り込みやROM-BIOSに予約することは避け

〈図1-3〉MS-DOSのメモリ・レイアウト



るべきです。

PC9801もIBM-PCもこの範囲はシステム予約となっています。

CP/M86とMS-DOSはよく比較されます。実際、MS-DOSのシステム・コール(INT 21h)の若い番号のパラメータの形式はCP/M86のそれとほとんど同じであり、CP/M86用に書かれたソフトウェアをMS-DOSに移すのは容易です。実際、CP/M86と初期のMS-DOS(V1.X)は機能的な差はあまりありません。

しかし、その後の拡張でCP/Mの機能を越えるインターフェース、例えばpathnameによるファイルの操作やストリームとしてのファイルの取り扱いが取り入れられ、CP/Mとの互換性はしだいに薄くなってきました。むしろ新たに取り入れられたシステム・コールの形式は、UNIXのそれに準じています。

## ②MASMによるプログラミング

MS-DOSで本格的に何かをやるとうる場合は、好むと好まざるとにかかわらずアセンブラでのプログラムを避けて通ることはできません。特に、FORTRANのようなアセンブラの記述の全くできない高級言語を使って、直接ハードウェアを操作したりするときはなおさらです。そのためのユーティリティとして、MS-DOSにはMASM(マクロ・アセンブラ)という強力なアセンブラが用意されています。

このMASMには、一般のアセンブラの機能に加えて、

- ▶マクロ機能を持つ
- ▶再配置可能(リロケータブル)なオブジェクトが得られる
- ▶多くの擬似命令が用意されている

などの便利な機能を持っています。また、出力されるオブジェクト・ファイルはMS-FORTRANなどのマイクロソフト社の他的高级言語のオブジェクト・ファイルと互換性を持ち、MS-LINKによってリンクすることも可能です。さらに、オブジェクト・ファイルはMS-LIBによって汎用性のあるライブラリとすることも可能です。

MS-DOSにはこのような便利なアセンブラがユーティリティとして付いていますが、大規模なプログラムを組む場合、いかにMASMでも大変です。したがって、一般的なプログラミングでは必要な部分をMASMで作り、それをMS-FORTRANなど的高级言語とリンクして使うという方法がとられます。しかし、MS-FORTRANなど的高级言語でMASMのルーチンを使用するときには面倒な制約がいろいろとあります。

そこで、ここではMS-FORTRANにおけるアセンブリ・ルーチンの使用を前提としてのMASMでの簡単なプログラムの組み方、およびMS-LINK、MS-LIBの使用法について説明し、その後、実際の使用例としてグラフィックなどのライブラリを作成することになります。

なお、以後よく出てくる用語について、いくつかを簡単に説明します。

#### ＜モジュール＞

個々に分離したコードの集合体を指します。リロケートابل・モジュールや実行可能モジュールなどがあります。コンパイラ(MASM、MS-FORTRANなど)が作成するオブジェクト・ファイルは、リロケートابل(再配置可能)モジュールで、絶対アドレスをその中に含まません。MS-LINKなどのリンクで、リロケートابل・モジュールをリンクすることにより、実行可能モジュールが作成されます。

以後モジュールといえば、リロケートابل・モジュール(オブジェクト・ファイルからドライブの指定および拡張子を取り除いた部分)を指します。

#### ＜外部参照＞

別のモジュール内にあるルーチンや変数を参照することを意味します。

#### ＜プロシジャ(手続き)＞

サブルーチンと考えてください。最後にはRETがいます。

それでは、MASMを使ったマクロ・アセンブラのプログラムの組み方を、擬似命令、入出力を中心に説明していくことにしましょう。

なお、ここでは8086のアセンブラのプログラムに関する説明はしません。8086のアセンブラについての基本的な知識があるものとして話を進めます。

MASMでは、名前(アドレス、データ、定数など)として以下のキャラクタが使用できます。

A～Z, a～z, 0～9, ?, @, -, \$

ただし、最初のキャラクタは英字でなければならず、また31文字を超える文字は無視されます。

では、まずMASMでのプログラムを最も特徴づけている擬似命令について説明します。ただし、ここでは、あくまでMS-FORTRANからの呼び出しを前提としてプログラミングを考えているので、それに必要と思われる最低限の命令のみを説明することにします。なお詳しい使用例、プログラミング例は、後に出てくるリストを参照してください。

まず初めに、擬似命令について説明します。

擬似命令(ディレクティブ)とは、それ自体はマシン語のコードに変換されませんが、アセンブラに対し各種の情報を与える命令です。擬似命令にはメモリ擬似、マクロ擬似、条件擬似、リピート擬似、リストイング擬似などがあります。

## 2-1 メモリ擬似命令

この擬似命令グループは、主にメモリの制御に関する情報をアセンブラに送ります。ただし、厳密に言えばこのグループは、いわば「その他」のグループで、COMMENTのようなメモリの制御に関係のない命令も含みます。このうち重要と思われるものについて説明します。

### ① <名前> SEGMENT <属性>

```
    {  
    <名前> ENDS
```

この擬似命令SEGMENTとENDSには含まれたプログラムやデータにおけるセグメントに名前を付けます。プログラムやデータのセグメントには、必ずこの擬似命令によって名前が付けられていなければなりません。

<属性>はアライン、組み合わせ、クラスです。クラスはシングルのクォート「'」で囲まれていなければなりません。これらの情報はリンクに渡されます。詳細はMS-LINKのところで説明します。

### ② <名前> GROUP <セグメント名> [, ...]

GROUPは、いくつかのセグメント名(SEGMENTによって名前が付けられていなければなりません)をまとめて1つの名前で参照できるようにします。GROUPは全体で64Kバイトを超えてはいけません。

### ③ ASSUME <レジスタ> : <名前> [, ...]

セグメント・レジスタがどのセグメント名を指すかをアセンブラに告げます。セグメント・レジスタは四つ(CS, DS, ES, SS)あり、ユーザはASSUMEに対して、一つから四つのレジスタを指示できます。



DATA	SEGMENT 'DATA'	データ部	
	〜	(セグメント名DATA, クラス名DATA)	
DATA	ENDS		
DGROUP	GROUP DATA	DGROUPにまとめる。	
CODE	SEGMENT 'CODE'		
	ASSUME CS:CODE,DS:DGROUP,SS:DGROUP	ASSUMEでセグメント指定	
	PUBLIC PROC1	PROC1 が外部参照できる様にする。	
PROC1	PROC FAR	外部参照されるプロシジャ	コード部
	〜	プロシジャ本体	(セグメント名CODE クラス名CODE)
	RET		
PROC1	ENDP	プロシジャ名 PROC1	
	〜	他のプロシジャ群	
CODE	ENDS		
	END	プログラムの終り	

#### ④ EXTRN <名前> : <タイプ> [, ...]

EXTRNの後に続く<名前>(他のモジュール内で定義されていること)が外部参照であることを定義します。このとき<名前>は、それが定義されているモジュール内でPUBLICによって外部参照の宣言がされていなければなりません。

<タイプ>は、<名前>がラベルやPROC擬似命令で定義されたプロシジャ名のとき、NEAR, またはFARを用います(NEARは省略可)。<名前>が同じセグメント内で定義されている(参照する側もされる側も同じセグメント内にある別個のモジュールに同じセグメント名を付けた場合)ときは、EXTRN擬似命令はSEGMENT〜ENDS内に書き、<タイプ>はNEARを用います。

また、<名前>が異なるセグメント内で定義されているときは、EXTRN擬似命令をSEGMENT〜ENDSの外に置き、<タイプ>はFARを用います。

<名前>が変数のときは、<タイプ>は変数のサイズを表し、次のいずれかです。

BYTE (1バイト)  
WORD (2バイト)  
DWORD (4バイト)

#### ⑤ PUBLIC <名前> [, ...]

<名前>が、他のモジュールで参照できるよう宣言します。

<名前>は、数、変数、ラベル、プロシジャ名のいずれかです。ただし、<名前>はレジスタ名あるいはEQUにより浮動小数点、数値、あるいは2バイトを超える整数値に定義されてはいけません。

#### ⑥ <名前> PROC [NEAR] または FAR { ENDP

この擬似命令にはさまれた一連の手続きに名前を付けます。また、この手続きの最後にはRETが必要です。

手続きが同じセグメント内から呼び出される場合はNEARを使い、異なるセグメントのモジュールから呼び出される場合はFARを使います。省略された場合はNEARとなります。

また、PUBLIC擬似命令と組み合わせることにより、FORTRANなどの他の高級言語からの呼び出しを受けることができます。

#### ⑦ ORG [<式>]

ロケーション・カウンタを<式>の値にセットします。コードはその値で始まる番地から生成されます。

#### ⑧ END [<式>]

ソース・ステートメントの終わりを示します。<式>があった場合、その値がプログラムの開始番地になります。ただし、いくつかのモジュールをリンクする場合、メイン・モジュールのみが<式>を用いることができます。

以上の擬似命令を使えば、MS-FORTRANとリンクする場合のMASMのプログラムは、一般的にリスト2-1のようになります。

### 2-2 リンク時の注意点

MASMのプログラムをMS-FORTRANなどとMS-LINKを使ってリンクする場合、以下のことに注意

①アセンブラ・ルーチンでコードを配置するセグメントには、通常は必ずCODEと名前を付けてください。また、データを配置するセグメントは名前をDATA、クラス名を'DATA'とし、DGROUPという名前のグループ内に置きます。またASSUME文が必要です。これは以下の理由に基づきます。

MS-FORTRANは、メモリの低位アドレスから高位アドレスに向かってコード、DGROUP、名前付きCOMMONブロックと取っていきます。したがって、ユーザが勝手に名前を付けたセグメントにあるアセンブラ・プログラムとMS-FORTRANなどをリンクしたとき、FORTRANがアセンブラ・プログラムを破壊することがあります(FORTRANで名前付きCOMMONブロックを定義した場合)。

③FORTRANなどから参照されるアセンブリ・ルーチンはPROC擬似で名前を付け、その名前をPUBL

### 2-3 条件疑似命令

すべての条件擬似命令は以下の書式にしたがいます。

```

    {
ELSE
    {

```

この命令があった場合、アセンブラは条件が真のときIFからELS文までのソースをアセンブルし、偽のときはELS文以下のソースをアセンブルします(ELS文がないときは条件が偽のとき条件文全体が無視されます)。

この擬似命令が、MASMを最も特徴づけている命令です。

この擬似命令を使えばソース・プログラムがすっきりして見やすくなります。ただし、サブルーチンと異なり常にコードを生成するため、あまり多用するとソース・プログラムが小さくてもオブジェクト・モジュールが巨大になってしまいます。ありがたがるのもほとんどにしましょう。

では、以下にマクロ擬似命令において重要と思われる

呼び出し部



る事柄について説明します。

### ① <名前> MACRO [<ダミー>, ...]

ENDM

擬似命令MACROとENDMには含まれたブロックに名前を付け、その名前をマクロ・コールできるように定義します。展開時にブロック内を部分的に変更したい場合は、その部分にダミーの名前を与えその名前をMACROの後に並べなければいけません。

マクロ・コールは以下のようにしてなされます。

<名前> [<パラメータ>, ...]

パラメータはMACROのダミーと一対一に対応します。アセンブル時にはパラメータの内容がダミーに順次置き換えられます。詳しくはリスト2-2および後のリストを参照してください。

### ② LOCAL [<ダミー>, <ダミー>, ...]

マクロ・ブロック内でラベルを使う場合、そのままでは2回以上展開したとき、同じラベルが使われるのでラベルの2重定義となってしまう。そこで、そのようなときにラベルをLOCAL擬似命令で局所的なラベルと定義すればアセンブラは展開時にそのラベルを独自のラベルに置き換えてくれます。

なお、このLOCAL擬似命令はマクロ・ブロック内の先頭に書かなければいけません。

## 2-5 リピート擬似命令

この命令は、同じ記述を何度も繰り返して使いたいときに用います。ただし、マクロと違って定義した場所で展開されます。ここでは簡単な説明にとどめることにします。

### ① REPT <式>

}

ENDM

これらの擬似命令には含まれたブロックを<式>の数だけ繰り返します。<式>は、16ビットの整数値として評価されます。

擬似命令には他にも多くの命令がありますが、ここでの説明は省きます。これ以上の説明はマクロ・アセンブラのマニュアルを見たほうが早いし正確でしょう。

それではMASMの擬似命令に関する説明はこれくらいにして、次にMASMの実行方法について簡単に説明し、その後FORTRANとリンクするうえで最も重要な入出力について説明することにします。

## ③MASMの実行方法

MASMは、リスト3-1に示すようにMASMを実行後にそれぞれのプロンプトに対して必要な入力を与えることで実行できます。

なお、コマンド・キャラクタとして「; (コロン)」と「CTRL-C」があります。これらはどこにおいても使用でき、「;」は次からのプロンプトに対しリターン・キーのみが入力されたときと同じように実行されます。CTRL-Cは、実行を中断させます。これらのコマンド・キャラクタについては、MS-LINK、MS-LIBでも同様に使えます。

MASMによりMS-FORTRANなどのサブルーチンや関数を作成する時に、FORTRANからアセンブラ・ルーチンへ数値、文字などを引き渡したり、逆にアセンブラ・ルーチンからFORTRANの方へ数値などを戻したりする必要があります。このFORTRANとアセンブラ・ルーチンとの間のインターフェースについて次に説明します。

### 3-1 変数、定数の型とその内部表現

FORTRANとアセンブラとの引数の受け渡しにおいては、その変数の型と内部表現について考慮する必要があります。そこで、MS-DOS上の代表的なFORTRANとして、MS-FORTRANとPC-

〈リスト3-1〉MASMの実行列

A)MASM

The Microsoft MACRO Assembler

Version 1.20, Copyright (C) Microsoft Inc. 1981,82,83

Source filename (.ASM): GRAPH ソースファイル名。

Object filename (GRAPH.OBJ): 出力するオブジェクトファイル名 (省略するとソースファイル名)。

Source listing (NUL.LST): 出力するソースリストファイル名 (省略するとなし)。ディスプレイに表示するときは、CON:

Cross reference (NUL.CRF): 出力するクロスリファレンスファイル名 (省略するとなし)。

あるいは、

A)MASM GRAPH:

FORTRANについてそこで使用される変数、定数の型とその内部表現について以下にまとめます。

### ①整数型

整数型には、データ長が2バイトの**2バイト整数型**と4バイトの**4バイト整数型**があり、両方とも内部表現は2の補数によって表されます。

変数のメモリへの格納状態は、図3-1のように1バイト(8ビット)ごとに低位バイトから順次高位のバイトが高位アドレス側へとなるように格納されています。

### ②実数型

実数型は、データ長が4バイトの**実数型**と8バイトの**倍精度実数型**があります。内部表現は、MS-FORTRANではバージョン3.0より後はIEEE型の内部表現ですが、3.0以前ではマイクロソフト社の実数フォーマットですので注意が必要です。ただし、この一方のフォーマットから他方のフォーマットへの変換はライブラリでサポートしています。

PC-FORTRANでは**／87スイッチの有無**で内部

表現が異なります。／87スイッチなしの場合、BASIC型の内部表現となり、／87スイッチありの場合にはIEEE型の内部表現となります。この2つの型の変換もライブラリでサポートされていますのでマニュアルを参照してください。

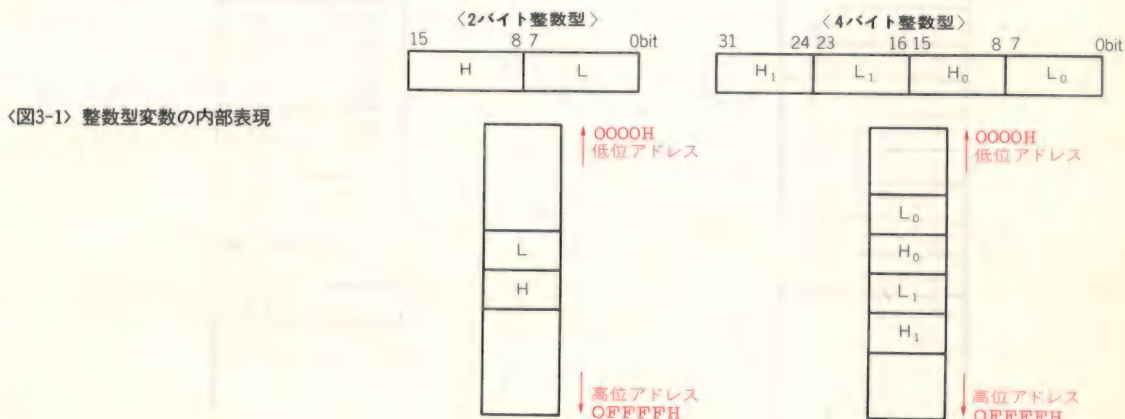
内部表現は浮動小数点形式で、指数部と仮数部からなっていて、データの値と実際のビットとの関係は図3-2のようになっています。

メモリへの格納は、整数型と同様に1バイトごとに低位のバイトから順次高位のバイトが高位アドレス側へとなるように格納されていきます。

### ③論理型

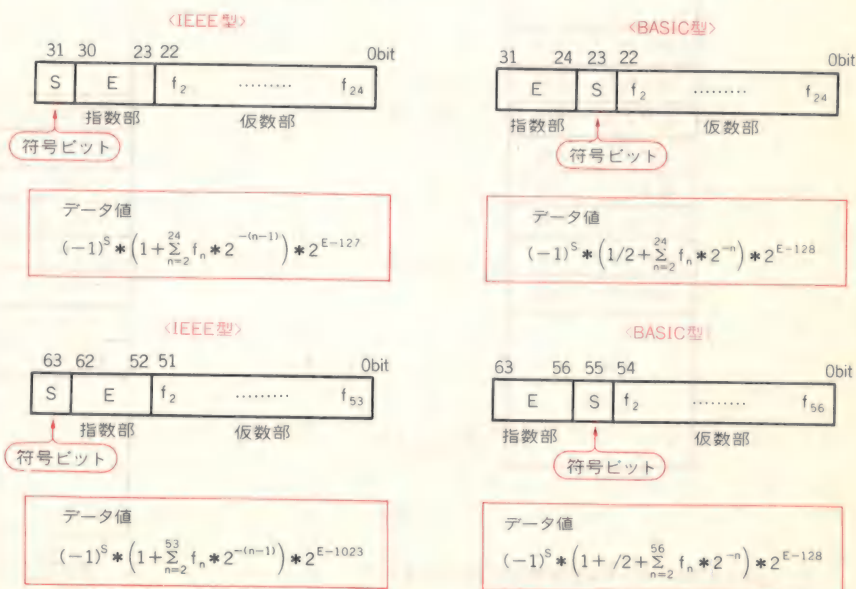
論理型変数のデータ長、内部表現は、MS-FORTRANとPC-FORTRANでは異なっています。MS-FORTRANではデータ長は2バイトか4バイトで、内部表現は偽のときは下位バイトが0、真のときは1となり、上位バイトは不定となります。

PC-FORTRANではデータ長が1バイトか4バイト



〈図3-1〉 整数型変数の内部表現

〈図3-2〉 実数型変数の内部表現





トで、偽のときは0を、真のときは0以外の値をとります。

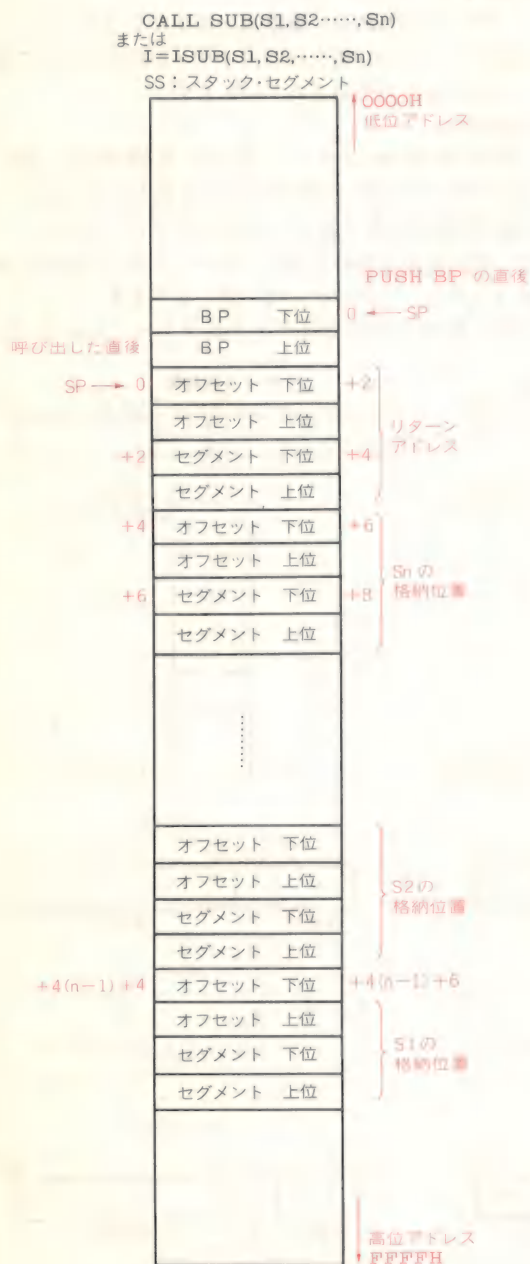
#### ④文字型

文字型のデータ長は指定された長さ分だけとられます。また、内部表現はASCIIコードで表されます。格納状態は最初の文字が低位アドレス側へと入り、順次高位アドレス側へと格納されていきます。

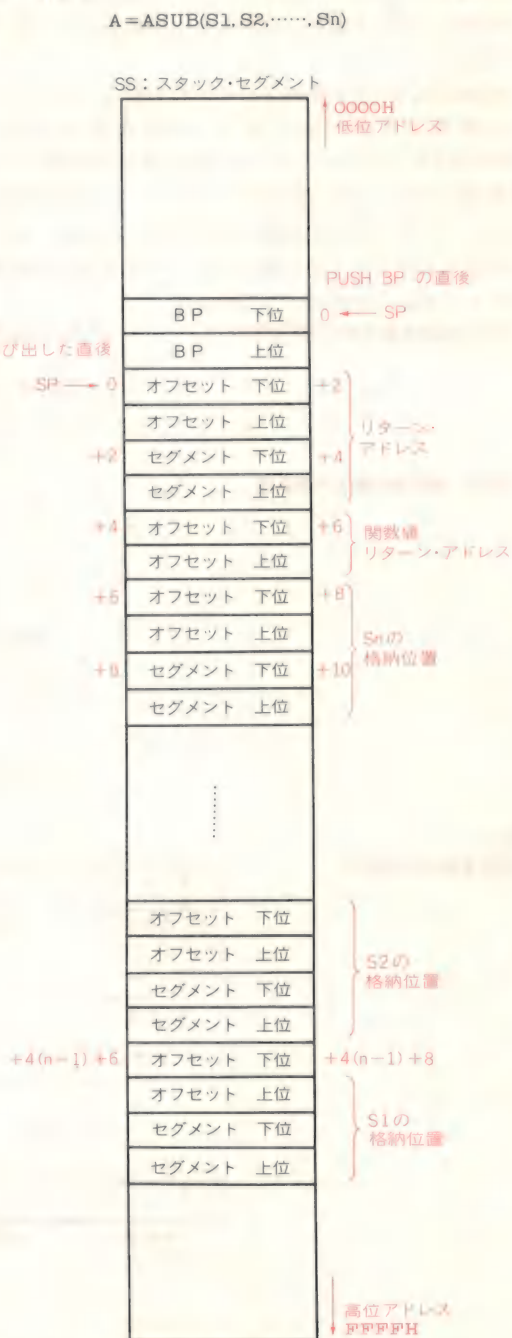
### 3-2 引数および値の受け渡し

MS-FORTRANでは、サブルーチンあるいは関数を呼び出したとき、そのルーチンを実行する直前に引数や戻り値などのセグメントやオフセットをスタックに積みます(戻り値はオフセットのみ)。このセグメントとオフセットを使えば、引数の値をレジスタやメモ

〈図3-3〉 スタックの状態



(a) サブルーチンおよび整数型ファンクション



(b) 実数型ファンクション

りに転送したり、逆にレジスタやメモリから引数へと転送したりできます。

また、関数での戻り値は、2バイト整数型ではAXレジスタ、4バイト整数型では下位2バイトがAXレジスタ、上位2バイトがDXレジスタで示されます。実数型の関数で結果を返すときは、スタックにそれ用

### リスト3-2 サブルーチンのサンプル・プログラム

```

COMMENT *
      ポート 出力 サブルーチン
SUBROUTINE OUT(ADDRESS, DATA)
ADDRESS : 2 バイト 整数型
DATA    : 2 バイト 整数型

*
DATA    SEGMENT 'DATA'
DATA    ENDS
DGROUP GROUP DATA
CODE    SEGMENT 'CODE'
ASSUME  CS:CODE, DS:DGROUP, SS:DGROUP
:
PUBLIC  OUT
PROC    FAR
PUSH    BP
MOV     BP, SP
:
LES     SI, [BP]
MOV     DX, ES:[SI]
LES     SI, [BP+2]
MOV     AX, ES:[SI]
OUT     DX, AL
:
POP     BP
RET     6
OUT     ENDP
CODE    ENDS
END

```

データ部、セグメント名DATA、クラスをDATA  
DGROUPにまとめる  
コード部、セグメント名CODE、クラスをCODE  
CS:CODE, DS:DGROUP, SS:DGROUP  
OUTを外部参照できるようにする  
プロシジャ名OUT、タイプFAR  
BPを保存する  
BPにSPの値をロードする  
第1引数の格納アドレスのセグメントをESに、オフセットをSIにロード  
第1引数の値をDXにロード  
第2引数の値をAXにロード  
ポートに出力  
BPを復元する  
6バイトカススタックを払い、リターンする

A>TYPE BEEP.FOR サブルーチンOUTの使用例  
DO 20 I=0,2000  
20 CALL OUT(837,6)  
CALL OUT(837,7)  
STOP  
END

### リスト3-3 整数関数のサンプル・プログラム

```

COMMENT *
      ポート 入力 関数
FUNCTION IN(ADDRESS)
ADDRESS : 2 バイト 整数型

*
DATA    SEGMENT 'DATA'
DATA    ENDS
DGROUP GROUP DATA
CODE    SEGMENT 'CODE'
ASSUME  CS:CODE, DS:DGROUP, SS:DGROUP
:
PUBLIC  IN
PROC    FAR
PUSH    BP
MOV     BP, SP
:
LES     SI, [BP]
MOV     DX, ES:[SI]
IN     AL, DX
XOR     AH, AH
XOR     DX, DX
:
POP     BP
RET     4
IN     ENDP
CODE    ENDS
END

```

第1引数の値をDXにロード  
ALにデータ出力する  
AX,DXを0にする、AX,DXが関数の戻り値となる  
リターン4バイト・ポップ

のオフセットが積んでありますから、そのオフセットの示すアドレスに戻り、値を書き込んでリターンすれば結果が返ります。

逆に、アセンブラ・ルーチンからFORTRANのサブルーチンなどを参照したいときは、スタックに引数の値が収まっているアドレスのセグメントとオフセット(それぞれ4バイト)、次に結果を入れるアドレスのオフセット(実数関数を呼び出すとき2バイト)を積んでコールします。

実際にサブルーチンと関数が呼ばれた後のスタックの状態を図3-3に示します。また、以上のことに関するサンプル・プログラムをリスト3-2～リスト3-5に示します。

リスト3-2はサブルーチンのサンプルです。このルーチンはポートに1バイトのデータを転送します。FORTRANからは、

**CALL OUT(IP, ID)**

(IP:ポート・アドレス, ID:転送データ)

のように呼び出します。ただし、データは2バイト整数IDの下位に入れます。なお、付属のFORTRANプログラムはOUTのテスト・プログラムです。実行

### リスト3-4 実数関数のサンプル・プログラム

```

COMMENT *
      実数関数 サンプルプログラム
      FUNCTION EQ(X)
      引き数の値をそのまま返す

*
DATA    SEGMENT 'DATA'
DATA    ENDS
DGROUP GROUP DATA
CODE    SEGMENT 'CODE'
ASSUME  CS:CODE, DS:DGROUP, SS:DGROUP
:
PUBLIC  EQ
PROC    FAR
PUSH    BP
MOV     BP, SP
:
LES     DI, [BP+8]
MOV     BX, [BP+6]
:
MOV     AX, ES:[DI]
MOV     [BX], AX
MOV     AX, ES:[DI+2]
MOV     [BX+2], AX
:
MOV     AX, [BP+6]
POP     BP
RET     6
EQ     ENDP
CODE    ENDS
END

```

第1引数の格納アドレスのロード  
戻り値の格納されるオフセットをBXにロード  
引数の下位2バイトをBXのアドレスにロード  
引数の上位2バイトをBX+2のアドレスにロード  
AXに戻り値の格納されているオフセットをロード

C 実数関数 サンプルプログラム  
C EQ TEST PROGRAM  
C  
A=0.0  
B=14.5  
WRITE(\*,5) A,B  
A=EQ(B)  
WRITE(\*,5) A,B  
STOP  
5 FORMAT(1H ,2F10.5)  
END



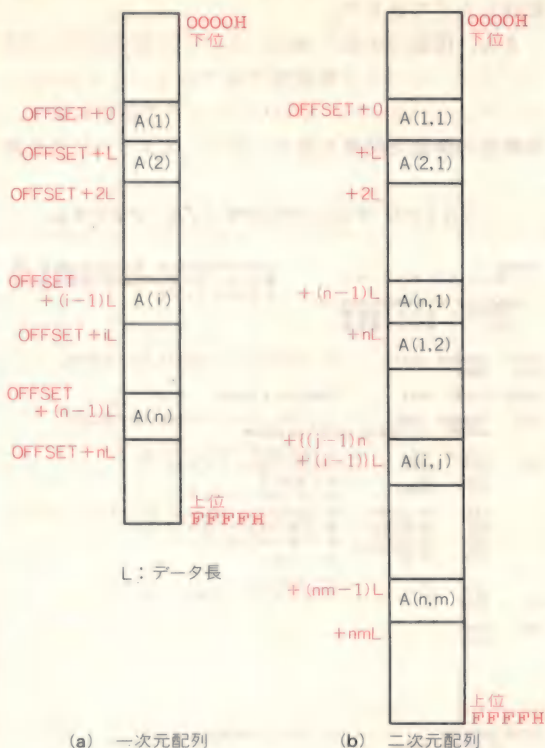
### 〈リスト3-5〉 アセンブラからFORTRANを呼び出す サンプル・プログラム

```

COMMENT *
FORTRAN FUNCTION CALL
*
;
; DGROUP GROUP DATA
;
; EXTRN AMUL:FAR          ← FORTRANの関数AMULを外部参照宣言する。タイプはFAR、
; CODE SEGMENT 'CODE'    ← SEGMENT擬似の外に書く
; ASSUME CS:CODE, DS:DGROUP, SS:DGROUP
;
; PUBLIC TEST
; PROC FAR
; PUSH BP
; MOV BP, SP
;
; LES DI, [BP+14]          ← 第1引数
; PUSH ES
; PUSH DI                  ← セグメントとオフセットをスタックに積む
;
; LES DI, [BP+10]          ← 第2引数
; PUSH ES
; PUSH DI
;
; MOV AX, OFFSET DS:FNSTACK ← DSからのオフセットFNSTACK
; PUSH AX                  ← 戻り値のオフセットをスタックに積む
;
; CALL AMUL                ← AMULをコール
;
; MOV SI, AX                ← 戻り値のオフセットをSIにロード
; LES DI, [BP+6]           ← 第3引数
;
; MOV AX, [SI]              ← 第3引数に結果を入れる
; MOV ES:DI, AX
; MOV AX, [SI+2]
; MOV ES:DI+2, AX
; POP BP
; RET 12                    ← リターン12バイト・ポップ
;
; TEST CODE ENDP
; ENDS
;
; DATA SEGMENT PARA 'DATA'
; FNSTACK DW 0              ← 関数の戻り値の格納領域
; DW 0
; DATA ENDS
;
; C
; C FORTRAN FUNCTION      MS-FORTRANの関数
; C
; C FUNCTION AMUL(A,B)
; C AMUL=A*B
; C RETURN
; C END
;
; C SAMPLE TEST PROGRAM   テスト・プログラム
; C
; C A=2.0
; C B=1.5
; C C=B
; C CALL TEST(A,B,C)
; C WRITE(*,*) A,B,C
; C STOP
; C END

```

〈図3-4〉 配列変数の格納のされ方



リスト3-5はアセンブラからFORTRANの関数を呼び出すサンプルです。このプログラムは、アセンブラ・レベルで実数の掛け算をするルーチンを作ろうとするものです。これをすべて8086のアセンブラでやろうとするとやたら手間がかかります(8087が使える人は別ですが)。そこでFORTRANの関数を拝借すれば、アセンブラ・レベルで簡単に実数計算ができるというわけです。

### 3-3 実数型関数を作るときに注意

通常、呼び出されたサブルーチンや関数では、DS、BP、SSの各レジスタは保持する必要がありますが、その他のレジスタは保持する必要がありません。ところが実数型関数の場合は、MS-FORTRANのところで、AXレジスタに、関数の戻り値が入るところのオフセット・アドレス(すなわち [BP+6])を入れてから関数のルーチン呼び出しをしています。そのため、関数のルーチン内でAXレジスタを変えたりすると、結果としてとんでもない値が返ることになります。

したがって、関数内でAXレジスタを変更した場合は、POP BPの前に、

```
MOV AX, [BP+6]
```

を入れる必要があります。

させると98が鳴きます。

リスト3-3に示したのは整数関数の例です。この関数はポートから1バイトのデータを読み込みます。FORTRANでは、

```
ID = IN(I,P)
```

のように使います。データは2バイト整数IDの下位に入り、上位バイトは0になります。

これらOUT、INのルーチンは有用なのでライブラリとしておくとよいでしょう。

リスト3-4は実数関数のサンプルです。これは引数の値をそのまま結果として返す関数です。実にしようもないプログラムですが、関数の作りは理解できると思います。

〈図3-5〉 MS-FORTRANのメモリ構造



DGROUP

DGROUPデータ領域		
セグメント	クラス	内 容
HEAP	MEMORY	ポインタ変数、いくつかのファイル
MEMORY	MEMORY	使用されない
STACK	STACK	フレーム変数およびデータ
DATA	DATA	静的変数
COMADS	COMADS	名前付きCOMMONブロックのアドレス
CONST	CONST	定数データ
COMMQQ	COMMON	名前なしCOMMONブロック

### 3-4 配列変数の配置と受け渡し

一次元配列変数は、メモリ上に添字1からDIMENSIONなどで定義された最大値まで順に格納されていて、一つの変数のデータ長は変数の型によるデータ長だけとります。例えば、

CALL SUB(A)

または、CALL SUB(A(1))

(SUB: サブルーチン名, A: 配列名)

とすると、A(1)の格納位置を示すセグメントとオフセットがスタックに積まれます。また、

CALL SUB(A(k))

とすると、A(k)のアドレスがスタックに積まれます。このアドレスからのオフセットを用いることで、配列の各要素の参照ができます(図3-4参照)。

二次元配列変数についても図3-4を参照してください [A(n, m)のとき]。

### 3-5 MS-FORTRANのメモリ構造

ここでいきなりですが、MS-FORTRANのメモリ構造について説明します。FORTRANのメモリ構造を知っておいたほうが、以降の事柄が理解しやすいからです。

MS-FORTRANのメモリ構造は、図3-5のようになっています。FORTRANの定数、変数スタック、ヒープ、無名共通ブロック、および名前付き共通ブ

ックのアドレスは、このうちのDGROUPというGROUPに割り当てられています。このDGROUPの中でメモリはオフセットFFFFhから順次低位に配置されます。したがって、DGROUPの最下位に配置された1バイトのオフセットは0以上の正のオフセットとなります。

DGROUPの中で無名共通ブロックは、セグメント名がCOMMQQ、クラス名COMMONのセグメントに配置されます。また、名前付き共通ブロックは、そのオフセット・アドレス、セグメント・アドレスがセグメント名COMADS、クラス名COMADSのセグメントに配置されます。実際には、名前付き共通ブロックは、DGROUPのすぐ上に配置されます。

### 3-6 COMMONブロックの参照

上記の知識を基にして、アセンブラからMS-FORTRANのCOMMONブロックを参照することができます。

名前なしCOMMONブロックを参照するときは、リスト3-6を参考にしてください。このプログラムは、INITをコールした後、タイマ割り込みがかかるたびにTIMERルーチンに飛ぶようにしてあります。COMMONブロックを参照することによって、いちいちFORTRANからアセンブラ・ルーチンをコールすることなしに引数の受け渡しができます。

COMMONブロックを使えば、このような一種の



＜リスト3-6＞ 無名共通ブロックのサンプル・プログラム

```

COMMENT #
無名共通ブロックサンプルプログラム

*
COMM*Q SEGMENT COMMON 'COMMON' ← 名前なしCOMMONブロック
      DW 0          セグメント名 COMM*Q
      DW 0          クラス名 COMMON
      DW 0          アライメント COMMON
      ENDS          (領域をMS-FORTRANとオーバーラップさせる)
;
DGROUP GROUP COMM*Q
;
CODE SEGMENT 'CODE'
      ASSUME CS:CODE, DS:DGROUP, SS:DGROUP
;
TIMER PROC
      DS ← タイマ割り込みルーチン
      PUSH AX
      PUSH BX
      PUSH CX
      PUSH DX
      レジスタの保存
;
      MOV AX, WORD PTR CODE:[DS_TBL] ← DSの値をロード
      MOV DS, AX
;
      MOV AH, 2CH
      INT 21H
      システム・コール (CH:時, CL:分, DX:秒)
      MOV BX, OFFSET DS: HOUR
      MOV BYTE PTR [BX], CH
      MOV BX, OFFSET DS: MIN
      MOV BYTE PTR [BX], CL
      MOV BX, OFFSET DS: SEC
      MOV BYTE PTR [BX], DH
      時分秒をCOMMON領域にセット
;
      MOV AL, 20H
      OUT 0, AL
      MOV AH, 3
      INT 1CH
      タイマ割り込み終了
;
      POP DX
      POP CX
      POP BX
      POP AX
      POP DS
      レジスタの復帰
      IRET
      ← DSを保存するための領域
DS_TBL DW 0
TIMER ENDS
CODE ENDS
;
CODE SEGMENT 'CODE'
      ASSUME CS:CODE, DS:DGROUP, SS:DGROUP
;
INIT PROC
      FAR ← 初期化ルーチン
      DS
      AX
      BX
;
      MOV AX, DS
      MOV WORD PTR CODE:[DS_TBL], AX
      DSをセーブ
;
      CLI
      XOR AX, AX
      MOV DS, AX
      MOV BX, 20H
      MOV WORD PTR [BX], OFFSET TIMER
      MOV AX, CS
      割り込みヘクタ (タイマ割り込み)
      0000: [0020] タイマ割り込み処理ルーチンのオフセット
      0000: [0022] タイマ割り込み処理ルーチンのセグメント
;
      AND OUT AL, 0FEH
      2, AL } タイマ割り込み以外をマスク
;
      POP BX
      POP AX
      POP DS
;
      INIT:
      OWARI PROC
      FAR ← 終了ルーチン
      PUSH AX
      PUSH BX
      PUSH ES
      IN AL, 2
      OR AL, 1
      OUT 2, AL
      CLI
      XOR AX, AX
      MOV ES, AX
      MOV BX, 20H
      MOV ES: [BX], 195DH
      MOV ES: [BX+2], 0FD80H
      STI
      POP ES
      POP BX
      POP AX
      IRET
      ENDP
      OWARI:
      CODE ENDS
      END

INTEGER=2 HOUR, MIN, SEC
COMMON HOUR, MIN, SEC
CALL INIT
DO 100 I=1, 100
100 WRITE(*, 1000) HOUR, MIN, SEC
CALL OWARI
1000 FORMAT(1H, 12, ':', 12, ':', 12)
STOP
END
MS-FORTRAN部
  
```

COMADSセグメント クラス名' COMADS'

オフセット

＜図3-6＞  
名前付きCOMMONブロックの  
アドレスの格納のされ方

	2	4	6	8
オフセット・アドレス	セグメント・アドレス	オフセット・アドレス	セグメント・アドレス	
2 バイト		2 バイト		
最初のCOMMONブロック のアドレス			2番目のCOMMONブロック のアドレス	

並列処理(もちろん時分割ですが)のようなことも簡単に  
できるわけです。

名前付きCOMMONブロックの場合は、各COM  
MONブロックのオフセット・アドレス、セグメン  
ト・アドレスが、セグメント名COMADS、クラス  
名COMADSのセグメントに図3-6のように格納さ  
れています。各COMMONブロックの構成要素の参  
照はそのアドレスからのオフセットを用います。なお、  
COMMONブロックの順序はリンクがリンク時に

会った順序にしたがいます。名前付きCOMMONブ  
ロックの参照についてはリスト3-7を参考にしてくだ  
さい。簡単なプログラムですが、名前付きCOMMO  
Nブロックのアドレスがどのように配置されているか  
がわかると思います。

4 MS-LINKとMS-LIBの使い方

MASMでソース・ファイルをアセンブルしてできた

トランジスタ技術  
SPECIAL

## 〈リスト3-7〉 名前付き共通ブロックのサンプル・プログラム

```

COMMENT %
名前付きCOMMONブロック サンプルプログラム

%
DATA SEGMENT 'DATA'
HIDEBU DB 'HI-DE-BU-' 111
TAWABA DB 'TA-BA-BA-' 111
ABESHI DB 'A-BE-SHI-' 111
DATA ENDS

COMADS SEGMENT COMMON 'COMADS' 名前付きCOMMONブロックのアドレス格納領域
DB 1 DUP(?) セグメント名COMADS クラス名COMADS
COMADS ENDS

; 1バイトの領域確保 (最初の名前付きCOMMON
; ブロックのオフセット・アドレスが格納されている)
DGROUP GROUP DATA,COMADS

CODE SEGMENT 'CODE'
ASSUME CS:CODE,DS:DGROUP,SS:DGROUP

COMMON MACRO DOFF,DSEG,DAT ；マクロ部
PUSH DS
MOV DI,WORD PTR DSEG(POINT) ；COMMONブロックのオフセットを取り出す
ADD DI,DOFF ；各要素のオフセットを出す
MOV AX,WORD PTR DSEG(POINT+2) ；COMMONブロックのセグメントを取り出す
MOV ES,AX
MOV SI,OFFSET DS:DAT
CX,8BH
22バイト
転送 ES:[SI] ←DS:[DI]
CLD
REP MOVSW
POP DS
ENDM

PUBLIC INIT
PROC FAR
COMMON 8,8,HIDEBU ；最初のCOMMONブロックの最初の要素
COMMON 22,8,TAWABA ；2番目のCOMMONブロックの最初の要素
COMMON 8,4,ABESHI
RET
ENDP
END

MS-FORTRAN%
%
C
C 名前付きCOMMONブロック サンプルプログラム
C
CHARACTER*22 HIDEBU,TAWABA,ABESHI
COMMON /COM1/HIDEBU,TAWABA
COMMON /COM2/ABESHI
C
HIDEBU="You have already died."
TAWABA="You have already died."
ABESHI="You have already died."
C
WRITE(*,*)HIDEBU
WRITE(*,*)TAWABA
WRITE(*,*)ABESHI
CALL INIT
WRITE(*,*)HIDEBU
WRITE(*,*)TAWABA
WRITE(*,*)ABESHI
C
STOP
END

```

オブジェクト・ファイルや、MS-FORTRANなどの他の高級言語によりコンパイルされたオブジェクト・ファイルはそのままでは実行できません。オブジェクト・ファイルを他のオブジェクトやライブラリ(特にFORTRANなどのコンパイラは入出力関係や関数などのサブルーチンをライブラリとして持っています)と結合させたりして、実行可能なファイルを作成する必要があります。そのためのユーティリティとしてMS-DOSには、MS-LINK(リンカ)が付いています。オブジェクト・ファイルを、MS-LINKでリンクすることで初めて実行可能なファイルができます。図4-1は、リンカの働きを模式的に表したものです。

### 4-1 リンカMS-LINKによるセグメントの配置

MASMは、SEGMENT擬似のアライメント、組み合わせ、クラスにより、リンカにセグメントの結合、配置に関する情報を渡します。

#### ①アライメント

アライメントは、セグメント(SEGMENT擬似命令によって指定された64Kバイト以内の連続した領域)をメモリ上のどのアドレスから開始するかをリンカに告げます。アライメントには以下の4種類があります。

**BYTE**：セグメントはメモリのどのバイトからでも開始できる。

**WORD**：セグメントはメモリの偶数アドレスから開始される。

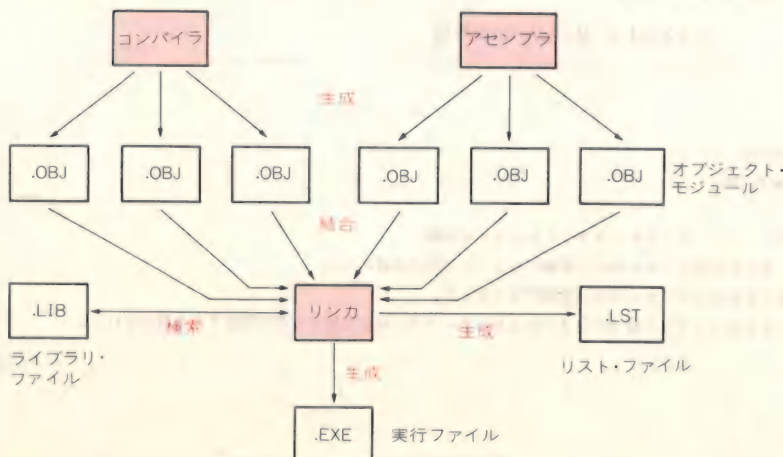
**PARA**：セグメントは下位4ビットが0のアドレスから開始される。

**PAGE**：セグメントは下位バイトが0のアドレスから開始される。

ここでMS-FORTRANやPC-FORTRANではセグメントはPARAに配置されることに注意してください。

#### ②組み合わせ

組み合わせは、特定のクラス名を持つセグメントの



〈図4-1〉 リンカの働き



配置方法をリンクに告げます。MS-LINKでは以下の4種類を扱います。

**指定なし(PRIVATE)**：セグメントは個別にロードされます。各セグメントが同じ名前、クラス名を持っている場合も、隣接してロードされますが、各セグメントは個別のベース・アドレスを持っています。

**PUBLIC**：同じ名前、クラス名のセグメントは隣接してロードされ、ベース・アドレスも1つです。オフセットはロードされた最初のセグメントから、ロードされた最後のセグメントまでです。

**STACK**：基本的にPUBLICと同じです。ただし、この指定をすると、ロードされた最後のセグメントの最後にスタック・ポインタが設定されます。これはスタック・セグメント用に使います。

**COMMON**：同じ名前、クラス名を持つセグメントをオーバーラップしてロードします。当然、ベース・アドレスは一つです。長さは最長セグメントの長さになります。

### ③クラス

同じクラス名を持つセグメントは連続してロードされます(違うセグメント名でも)。MS-LINKはオブジェクト・ファイル内でセグメントに出会った順番にロードします。

## 4-2 MS-LINKの実行方法

MS-LINKは、リスト4-1で示すようにLINKを実

行後にそれぞれのプロンプトに対して、必要な入力を与えることで実行できます。

コマンド・キャラクタとして、「**;(セミコロン)**」と「**CTRL-C**」の他に、「**+(プラス)**」が使えます。「**+**」は、オブジェクト・モジュールとライブラリのプロンプトのところで使えます。「**+**」は、**入力の区切り**として使う他に、**入力の終わりに付けるとプロンプトが繰り返される**ので、入力を数回に分けることができます。

FORTRANとアセンブラのプログラムとをリンクするときは、オブジェクト・モジュールの入力の際にはFORTRANのモジュールを先頭にして入力してください。さもないとセグメントの順序に狂いが生じることがあります。ただし、そのモジュールはメイン・モジュールである必要はありません。

### 4-3 MS-LIB(ライブラリ・マネージャ)

多くのプログラムで共通して使われる汎用性のあるサブルーチンは、ライブラリとしておけば便利です。また、特定のプログラムに対して専用のライブラリを作っておけば、リンクの速度も効率良くプログラムが開発できます。このためのユーティリティとして、MS-DOSにはMS-LIBが用意されています。

これを使えば、オブジェクト・ファイル単位にモジュール(ここでのモジュールは、オブジェクト・ファイルからドライブの指定子および拡張子を取り除いた部分のことで、オブジェクト・ファイル中の外部参照宣言されたプログラム・コード群からなっている)のライブラリへの追加、削除が容易にできます。MS-LIBには、必要最低限の機能しかついていませんが、通常はこれで十分だと思います。以下にMS-LIBの機能をまとめます。

① 新しいライブラリを作成できる。

② ライブラリへ新しいオブジェクト・ファイルをモジュールとして追加できる。

〈リスト4-1〉 MS-LINKの実行列

A)LINK

Microsoft Object Linker V2.2(830822)

(C) Copyright 1982, 1983 by Microsoft Inc.

Object Modules (.OBJ): EQTEST EQ ——— リンクするオブジェクトファイル群。

Run File (EQTEST.EXE): ——— 出力する実行ファイル名 (省略すると [ ] の中の名前になる)。

List File (NUL.MAP): ——— 出力するリストファイル名 (省略するとなし)。

Libraries (.LIB): ——— リンクするライブラリ名 (FORTRANのライブラリは自動検索なので指定する必要はない)。

あるいは、

A)LINK EQTEST EQ;

〈リスト4-2〉 MS-LIBの実行例

A)LIB ☐  
 Library file:GRAPH ☐ ① .LIB  
 Operation:+NEGA-CLCOPY\*PORT ☐ ② .OBJ 拡張子  
 List file:GRAPH.LST ☐ ③

それぞれの入力に際しては、右側に示した拡張子が自動的に決定される。

- ①ライブラリ名を入力する。新規のライブラリ名を入力した場合には、次に表示される  
 Library file dose not exist. Create?  
 に対して、Yを入力すればよい。  
 ②オブジェクトファイル名（ライブラリ中ではモジュール名）  
 にコマンドキャラクタを付加して入力する。  
 ③ライブラリ中のモジュールの外部参照シンボルのリストイン  
 グファイルが必要となきに入力する。

- ③ ライブラリから不必要なモジュールを削除できる。  
 ④ ライブラリからモジュールを引き出して、そのモジュールにしたがってオブジェクト・ファイルを作成できる。  
 ⑤ リスティング・ファイルを作成できる。  
 MS-LIBは、リスト4-2に示すように、LIBを実行後にそれぞれのプロンプトに対して必要な入力を与え

〈表4-1〉 コマンド・キャラクタの意味

コマンド・キャラクタ	意 味
+	次のオブジェクト・ファイルをモジュールとしてライブラリに追加する。
-	次のモジュールをライブラリから削除する。
*	次のモジュールをライブラリから抜き出し、オブジェクト・ファイルとして作成する。
&	コマンドが長くなるときに、モジュール、あるいはオブジェクト・ファイルの後に付けて入力するとプロンプトが繰り返されるので、コマンドを数回に分けて入力できるようになる。

ることで実行できます。

表4-1に、コマンド・キャラクタの種類と意味についてまとめました。

リスト4-2では、ライブラリGRAPH.LIBに新規のオブジェクト・ファイルNEGA.OBJを加え、GRAPH.LIBのモジュールCLCOPYを削除し、またモジュールPORTを抜き出しPORT.OBJなるオブジェクト・ファイルを作成することになります。また、例えばライブラリから元のモジュールを削除し、新規の同じ名前のモジュールを追加すると

好評発売中

別冊 **インターフェース**

# 数値演算プロセッサ

浮動小数点演算入門から高速演算プログラミングまで



CQ出版社

インターフェース編集部 編, B5判, 272頁定価1,800円 送料250円

- 第1部 パソコン数値演算入門と浮動小数点演算規格 IEEE 754 (4編)  
 浮動小数点演算入門/パソコン数値計算の誤差と対策/パソコンBASICの数学関数の誤差と対策/IEEE 754 浮動小数点演算規格の概要
- 第2部 数値演算プロセッサ 8087/80287の徹底活用 (8編)  
 8087 活用のための基礎知識/コンパイラでの8087 使用法/アセンブラでの8087 活用/CとFORTRANにおける8087 支援の評価/C言語における8087 インライン・サポート関数/高速・高精度行列演算のための8087 活用/8087を用いたFFTプログラム/マクロアセンブラを用いた8087 基本数学関数ライブラリ
- 第3部 IEEE 754 規格準拠各種数値演算プロセッサの研究 (5編)  
 MC68881/NS32081/μPD72191/ADSP3210,3220/WTL1164,1165



き(モジュールを更新するとき)には②において、次のようにします。

-NEGA+NEGA②

③は、ライブラリ中のモジュールのPUBLICシンボルのリスティング・ファイルが必要なときに入力してください。GRAPH.LSTという名前のリスティング・ファイルが生成されます。必要なければリターンのみ入力してください。

他の実行方法として、次のようにしてもかまいません。

> LIB <ライブラリ> <オペレーション>,  
<リスティング>

例えば、前の例と同じ処理を行うには、

> LIB GRAPH+NEGA-CLCOPY  
\*PORT, GRAPH.LST②

と入力してください。コマンド・キャラクタは前と同様に使用できます。

## ⑤MS-FORTRANで使えるユーティリティ

それでは、これまで説明してきたことの集大成として、MS-FORTRANで使えるグラフィックス、ハード・コピー、およびその他のユーティリティ・ライブラリを作成してみましょう。

### 5-1 GRAPH.LIBの作り方

まず、リスト5-1～リスト5-8を適当な名前で(例えばそれぞれ、GRAPH.ASM, CLCOPY.ASM, DIT5.ASM, DIT8.ASM, NEGA.ASM, MEMORY.ASM, MSSYS.ASM, BIT.ASM), エディタ(EDLINなど)で入力します。ファイル名拡張子は必ずASMとしてください。入力したソース・ファイルをMASMで以下のようにそれぞれアセンブルします。

```
A> MASM GRAPH:②
A> MASM CLCOPY:②
)
```

A> MASM BIT:②

できたオブジェクト・ファイル群をMS-LIBでGRAPH.LIBというライブラリにします。

A> LIB GRAPH②

```
Library file dose not  
exist. Create?Y②  
Operation:+GRAPH+CLCOP  
Y+DIT5+DIT8&②  
Operation:+NEGA+MEMOR  
Y+MSSYS+BIT②  
List file:GRAPH.LST②
```

これによって、GRAPH.LIB, GRAPH.LST(リスティング・ファイル)が生成されます。

GRAPH.LIBではグラフィック関係のルーチンにグラフィックLIOを使用しています。

これらリスト5-1からリスト5-8までは、単独でもライブラリとして使えます。ライブラリ中で用いられる引数は大部分2バイト整数ですが、ほとんどの場合は4バイト整数を使っても大丈夫です(一部異常動作するものがあります)。

GRAPH.LIBは、PC-FORTRANでも使える場合がありますが、なるべく使わないでください。PC-FORTRANではデータ部などの構造がMS-FORTRANとはまるで違うからです(DGROUPなるGROUPはありません)。ただし、一部リストを変更すれば動きます(DGROUPをとり、DSにDATAのセグメントを入れる)。

ハード・コピー関係のルーチンは、PC-PR201CL用に作りしました。他の機種でもリストを部分的に変更すれば動くと思います。

今回のGRAPH.LIBは、グラフLIOを使用したため、速度的にはN<sub>88</sub>BASICとほとんど変わりません。GDCを直接使用すれば、もっと速くかつ細かい制御ができます。余力のある人はやってみてください。

#### 参考文献

- (1) NEC, MS-DOS2.0各種マニュアル。
- (2) MICROSOFT, MS-FORTRAN USER'S GUIDE。
- (3) NEC, 日本語PC-FORTRAN SOFTWARE LIBRARY ユーザーズマニュアル。
- (4) NEC, PC-PR201CL 日本語カラーシリアルプリンタ, USER'S MANUAL。
- (5) 白田耕作; PC-98用拡張システム・コール詳説, インターフェース, 1985年4月号, pp.22~39, CQ出版社。
- (6) 別冊トランジスタ技術, ソフトマインドC on the PC98, 1985年, pp.175~199, CQ出版社。

```

1: COMMENT *
2: GRAPHICSROUTINE
3:   FOR PC-9801
4:   *
5: DGROUP GROUP DATA
6: ;
7: CODE SEGMENT 'CODE'
8:   ASSUME CS:CODE, DS:DCGROUP, SS:DCGROUP
9: ;
10: OFFSET MACRO PDA, SETD
11:   MOV SI, OFFSETDS: PDA
12:   SUB SI, BX
13:   MOV SETD, SI
14:   ENDM
15: ;
16: ;
17: PUBLIC GINIT
18: GINIT PROC FAR
19:   CLI
20:   PUSH DS
21:   MOV CX, 16
22:   CLD
23:   MOV AX, OF990H
24:   MOV DS, AX
25:   XOR AX, AX
26:   MOV ES, AX
27:   MOV SI, 6
28:   MOV DI, 0A0H*4
29: SET_VECTOR:
30:   MOVSW
31:   MOV AX, OF990H
32:   STOSW
33:   ADD SI, 2
34:   LOOP SET_VECTOR
35:   MOV DI, 0CEH*4
36:   MOVSW
37:   MOV AX, OF990H
38:   STOSW
39: ;
40:   MOV DI, 0C5H*4
41:   MOV AX, OFFSETSTOP_CHECK
42:   STOSW
43:   MOV ES: [DI], CS
44: ;
45:   POP DS
46:   STI
47: ;
48:   MOV AX, DS
49:   MOV BX, OFFSETDS:LIO_WORK
50:   REPT 4

```

### GINIT (グラフィックの初期化)

グラフィックL10の初期化を行う。グラフィックL10を使用する際には必ず最初にコールする必要がある。実行することで次のように設定される。

- ▶画面モードはSCREEN(3.0.0.r) (SCREENの項を参照) に設定される
- ▶ビデオポートはアダプティブ画面全体に設定される
- ▶画面は消去される。

←繰り返しベクタの設定

グラフィックL10を使用するときは、ワークエリアを確保しなければならないが、グラフィックL10はその先頭アドレスを、セグメントD S, オフセット0として参照する。そのためのDSをここで計算する。

```

51: SHR BX, 1
52: ENDM
53: ADD AX, BX
54: MOV DS, LIO, AX
55: ;
56: MOV BX, OFFSETDS:LIO_WORK
57: OFSET SCREEN_D, SC_LIO
58: OFSET PARAMETER, PA_LIO
59: OFSET WORK_START, WS_LIO
60: OFSET WORK_END, WE_LIO
61: ;
62: PUSH DS
63: ;
64: MOV AX, DS_LIO
65: MOV DS, AX
66: INT 0A0H
67: POP DS
68: MOV BX, SC_LIO
69: PUSH DS
70: MOV AX, DS_LIO
71: MOV DS, AX
72: MOV WORD PTR [BX], 0003H
73: MOV [BX+2], 0100H
74: INT 0A1H
75: INT 0A5H
76: ;
77: POP DS
78: RET
79: ;
80: STOP_CHECK:
81: IRET
82: ;
83: GINIT ENDP
84: ;
85: ;
86: ;
87: INIT MACRO DATA_D
88:   PUSH BP
89:   MOV BP, SP
90:   MOV BX, OFFSETDS:DATA_D
91:   ENDM
92: ;
93: INIT_NO_MACRO
94:   PUSH BP
95:   MOV BP, SP
96:   ENDM
97: ;
98: SUFB MACRO SB1, SB2
99:   LES SI, SB1[BP]
100:  MOV AX, ES:[SI]

```

オフセットを計算する。

←システム・コール(グラフィックL10の初期化)

←SCREEN 3.0.0.1  
←グラフィック画面の消去



101: MOV BYTE PTR S82[BX],AL

102: ENDM

103: ;

104: SUFW MACRO SW1,SW2

105: LES SI,SW1[BP]

106: MOV AX,ES:[SI]

107: MOV SW2[BP],AX

108: ENDM

109: ;

110: FIN MACRO EN,RN

111: PUSH DS

112: MOV BX,PA\_L10

113: MOV AX,DS\_L10

114: MOV DS,AX

115: ;

116: INT EN

117: ;

118: POP DS

119: POP BP

120: RET RN

121: ENDM

122: ;

123: PUBLIC SCREEN

124: SCREEN PROC FAR

125: INIT SCREEN\_D

126: SUFB 18,0

127: SUFB 14,1

128: SUFB 10,2

129: SUFB 6,3

130: ;

131: PUSH DS

132: MOV BX,SC\_L10

133: MOV AX,DS\_L10

134: MOV DS,AX

135: ;

136: INT OAH

137: ;

138: POP DS

139: POP BP

140: RET 16

141: SCREEN ENDP

142: ;

143: PUBLIC VIEW

144: VIEW PROC FAR

145: INIT PARAMETER

146: SUFW 26,0

147: SUFW 22,2

148: SUFW 18,4

149: SUFW 14,6

150: SUFB 10,8

### SCREEN (スクリーンの設定)

画面モード、ディスプレイ画面、アクティブ画面、ディスプレイ画面を設定する。

引数: J: 画面モード

K: ディスプレイ画面

L: アクティブ画面

このルーチンの呼び出しによって、アクティブ画面内の描画領域(ビューポート)はアクティブ画面全体となる。また、各パラメータの詳細および画面合成の可否についてはP-C-9801のマニュアルを参照のこと。

### VIEW (ビューポートの設定)

CALL VIEW(J1, J2, J2, IC1, IC2)

アクティブ画面内の描画領域(ビューポート)を指定する。またビューポート内の塗りつぶし、外枠の描画を行う。

(J1, J2) : (J2, J2)を対角線とする領域をビューポートとする。

引数: IC1: ビューポート内を塗りつぶす色

IC2: ビューポートの外枠の色

### 消去

GCLS: ビューポート内のグラフィック画面の消去

CLS: テキスト画面の消去

CLS3: グラフィックとテキスト画面の消去

### PSET (点を打つ)

CALL PSET(I, J, IC)

指定の座標にICの色で点を打つ。

引数: I: X座標

J: Y座標

IC: 描画色

### LINE (線を引く)

CALL LINE(I1, J1, I2, J2, IC)

(I1, J1)から(I2, J2)までICの色で線を引く。

```

201: LINE      PROC      FAR
202:   INIT      PARAMETER
203:   SUFW      22,0
204:   SUFW      18,2
205:   SUFW      14,4
206:   SUFW      10,6
207:   SUFB      6,8
208:   MOV      WORD PTR 9[BX],0000H
209:   FIN       0A7H,20
210: LINE      ENDP
211: ;
212: PUBLIC BOX
213: BOX        PROC      FAR
214:   INIT      PARAMETER
215:   SUFW      22,0
216:   SUFW      18,2
217:   SUFW      14,4
218:   SUFW      10,6
219:   SUFB      6,8
220:   MOV      WORD PTR 9[BX],0001H
221:   FIN       0A7H,20
222: BOX        ENDP
223: ;
224: PUBLIC BOXF
225: BOXF       PROC      FAR
226:   INIT      PARAMETER
227:   SUFW      26,0
228:   SUFW      22,2
229:   SUFW      18,4
230:   SUFW      14,6
231:   SUFB      10,8
232:   MOV      9[BX],0102H
233:   SUFB      6,11
234:   FIN       0A7H,24
235: BOXF       ENDP
236: ;
237: PUBLIC CIRCLE
238: CIRCLE     PROC      FAR
239:   INIT      PARAMETER
240:   SUFW      42,0
241:   SUFW      38,2
242:   SUFW      34,4
243:   SUFW      30,6
244:   SUFB      26,8
245:   SUFB      22,9
246:   SUFW      18,10
247:   SUFW      14,12
248:   SUFW      10,14
249:   SUFB      6,16
250:   FIN       0A8H,40

```

#### BOX (箱を描く)

CALL BOX(I1, J1, I2, J2, IC)  
(I1, J1) - (I2, J2)を対角線とする四角形をIC  
の色で描く。

#### BOXF (箱を塗りつぶす)

CALL BOXF(I1, J1, I2, J2, IC1, IC2)  
(I1, J1) - (I2, J2)を対角線とする四角形をIC1  
の色で描き、内部をIC2の色で塗りつぶす。

#### CIRCLE (精円または円を描く)

CALL CIRCLE(I1, J1, I2, J2, IC1, IC2, ISX, ISY, IEX, IEY)  
中心点(I1, J1), X方向半径RX, Y方向半径RY,  
ICの色で、精円または円の一部、または全部を  
描く。  
引数  
I1 : フラグ(bn: nビット)  
b0 : 開始点の座標(0:無し, 1:有)  
b1 : 開始点の座標(0:無し, 1:有)  
b2 : 終了点の座標(0:無し, 1:有)  
b3 : 終了点の座標(0:無し, 1:有)  
b4 : 開始点、終了点一致の時の描画方法  
指定(0:全精円を描画, 1:1点  
(一致点)のみ描画)  
b5~b7: 未使用

```

251: CIRCLE     ENDP
252: ;
253: PUBLIC PAINT
254: PAINT      PROC      FAR
255:   INIT      PARAMETER
256:   SUFW      18,0
257:   SUFW      14,2
258:   SUFB      10,4
259:   SUFB      6,5
260:   MOV      AX,WE_L10
261:   MOV      6[BX],AX
262:   MOV      AX,WS_L10
263:   MOV      8[BX],AX
264:   FIN       0A9H,16
265: PAINT      ENDP
266: ;
267: PUBLIC TILEP
268: TILEP      PROC      FAR
269:   INIT      PARAMETER
270:   SUFW      22,0
271:   SUFW      18,2
272:   SUFB      14,10
273:   MOV      AX,OFFSETS-TILE_D
274:   MOV      6[BX],AX
275:   MOV      AX,WE_L10
276:   MOV      16[BX],AX
277:   MOV      AX,WS_L10
278:   MOV      18[BX],AX
279:   LES      SI,6[BP]
280:   MOV      AX,ES:[SI]
281:   MOV      CX,AX
282:   SHL      AX,1
283:   ADD      AX,CX
284:   MOV      BYTE PTR 5[BX],AL
285:   MOV      8[BX],DS
286:   PUSH     BX
287:   LES      SI,10[BP]
288:   MOV      BX,OFFSETS-TILE_D
289:   T_LOOP:
290:   MOV      AX,ES:[SI]
291:   MOV      BYTE PTR 1[BX],AH
292:   MOV      BYTE PTR 2[BX],AL
293:   MOV      AX,ES:2[SI]
294:   MOV      BYTE PTR [BX],AL
295:   ADD      SI,4
296:   ADD      BX,3
297:   LOOP     T_LOOP
298:   POP      BX
299:   FIN      0AAH,20
300: TILEP      ENDP

```

#### PAINT (ペイント)

CALL PAINT(I1, J1, IC1, IC2)  
指定した開始点(I1, J1)と境界色IC2で決定さ  
れる領域をIC1の色で塗りつぶす。

#### TILEP (タイルング・ペイント)

CALL TILEP(I1, J1, IC, ITL, ID)  
指定した点(I1, J1)と境界色ICで決定される領  
域を指定のタイル・パターンITLで塗りつぶ  
す。  
引数  
I1 : 開始点のX座標  
J1 : 開始点のY座標  
IC : 境界色  
ID : タイル・パターン(4バイト整数)  
ITL : タイル・パターンの繰り返し数  
IDは最大10までとれる



```

301: ;
302: GET PUBLIC GET FAR
303: INIT PROC PARAMETER
304: 22,0
305: SUFW 18,2
306: SUFW 14,4
307: SUFW 10,6
308: SUFW 14,4
309: LES SL,6[BP]
310: MOV 8[EBX],SI
311: MOV 10[EBX],ES
312: ;
313: MOV AX,4[EBX]
314: SUB AX,[BX]
315: PUSH BX
316: ADD AX,8
317: XOR DX,DX
318: MOV BX,8
319: DIV BX
320: POP BX
321: MOV DX,AX
322: MOV AX,6[EBX]
323: SUB AX,2[EBX]
324: INC AX
325: PUSH BX
326: MOV CL,BYTE PTR SCREEN_D
327: CMP CL,0
328: JE COL1
329: CMP CL,1
330: JE COL2
331: CMP CL,2
332: JE COL2
333: COL1:
334: MOV BX,AX
335: SHL AX,1
336: ADD BX,AX
337: COL2:
338: MOV AX,DX
339: MUL BX
340: ADD AX,4
341: POP BX
342: MOV 12[EBX],AX
343: FIN OAH,20
344: GET END
345: ;
346: PUBLIC PUT FAR
347: INIT PROC PARAMETER
348: SUFW 30,0
349: SUFW 26,2
350:

```

### GET (画面情報を配列へ格納する)

CALL GET(IX1, IV1, IX2, IV2, I)

(IX1, IV1) - (IX2, IV2) を対角線とする領域の画面情報を配列に格納する。

引数

- IX1: 左上の X 座標
- IV1: 左上の Y 座標
- IX2: 右下の X 座標
- IV2: 右下の Y 座標
- I: 格納する配列 (2 バイト整数)

配列の大きさは、 $((IX2 - IX1 + 8) / 8) * (IV2 - IV1 + 1) * N + 4$  / 2 + 1 以上でなければならぬ。ここで N は、画面モードがカラーのとき 3 でモノクロのときは 1 である。

データ格納領域の計算  
A X ←

### PUT (配列へ格納した画面情報に描画する)

CALL PUT(IX, IV, I, M, S, IFC, IBC)

配列へ格納した画面情報を画面に描画する。

引数

- IX: 描画する左上の X 座標
- IV: 描画する左上の Y 座標
- I: 画面情報を格納してある配列
- M: 描画モード
- IS: カラー・スイッチ
- IFC: フォア・グラウンド・カラー
- IBC: バック・グラウンド・カラー

描画モード

- M = 0: PSET
- M = 1: NOT
- M = 2: OR
- M = 3: AND
- M = 4: XOR

### KPUT (グラフィック画面に日本語を描画する)

CALL KPUT(I, J, K, IC1, IC2, M)

グラフィック画面に日本語を描画する。

引数

- I: 描画する日本語の左上の X 座標
- J: 描画する日本語の左上の Y 座標
- K: 漢字コード
- IC1: 文字の色
- IC2: 文字以外の部分の色
- M: 描画モード

描画モードは PUT を参照のこと。  
文字はビューポート内にすべて入らなければならない。

### ROLL (画面スクロール)

CALL ROLL(I, J)

画面全体を指定ドット数だけ上下左右に移動させる。

引数

- I: 上下方向のスクロール・ドット数
- J: 左右方向のスクロール・ドット数

I が正の場合には、上方向、負の場合には下方向、I が正の場合には左方向、負の場合には右方向にスクロールさせる。-199 ~ 199 の範囲は、標準解像度で、-399 ~ 399、専用高解像度で、-639 ~ 639。

J は 8 の倍数である。8 の倍数でないときはその絶対値以下で最も近い 8 の倍数部だけスクロールさせる。

```

401: SUFW 6,2
402: MOV BYTE PTR 4[BX],00H
403: FIN OAEH,8
404: ROLL ENDP
405: ;
406: PUBLIC IPOINT
407: IPOINT PROC FAR
408: INIT PARAMETER
409: SUFW 10,0
410: SUFW 6,2
411: ;
412: PUSH DS
413: MOV BX,PA_L10
414: MOV AX,DS_L10
415: MOV DS,AX
416: ;
417: INT OAFH
418: ;
419: POP DS
420: XOR AH,AH
421: XOR DX,DX
422: POP BP
423: RET 8
424: ;
425: IPOINT ENDP
426: ;
427: MACRO NA,RS
428: LOCAL LOP
429: PUSH CX
430: PUSH BX
431: MOV BX,OFFSETDS:NA
432: MOV CX,RS
433: LOP:
434: PUSH CX
435: MOV CL,[BX]
436: CALL LPRINT
437: INC BX
438: POP CX
439: LOOP LOP
440: POP BX
441: POP CX
442: ENDM
443: ;
444: PUBLIC TCOPY
445: TCOPY PROC FAR
446: PUSH BP
447: ;
448: MOV YL,0
449: MODE PRINT12,5
450: LOOP_YLT:

```

IPOINT (ドット情報を取り出す)

IC=IPOINT(I, J)

指定位置(I, J)のドット情報を取り出す。  
ICにドット情報が入り、ICはカラー・モード  
時はその色が入り、モノクロ・モード時は  
は白の時1、黒の時は0になる。

これよりハード・コピー・ルーチン  
+ プリンタのモードの設定

TCOPY

テキスト画面のハード・コピーを取る。

COPY

CALL COPY(1)

グラフィック画面のみ、あるいはテキスト  
画面と合成してハード・コピーをとる。

1 : 0の時グラフィック画面のハード・コ  
ピー、1の時テキスト画面と合成した  
グラフィック画面の1ドットをプリンタの  
2ビット(1×2)として、横方向には1ビ  
ットずつスキップする。

DCOPY

CALL DCOPY(1)

グラフィック画面のみ、あるいはテキスト  
画面と合成してハード・コピーをとる。  
1はCOPYと同じ。

グラフィック画面の1ドットをプリンタの  
4ビット(2×2)とする。

```

451: CALL TCOPY_R
452: MOV CL,LF
453: CALL LPRINT
454: ;
455: INC YL
456: CMP YL,50
457: JB LOOP_YLT
458: ;
459: MODE PREND,5
460: ;
461: POP BP
462: RET
463: TCOPY ENDP
464: ;
465: ;
466: PUBLIC COPY
467: COPY PROC FAR
468: PUSH BP
469: MOV BP,SP
470: ;
471: LES SI,8[BP]
472: MOV AX,ES:[SI]
473: MOV BYTE PTR CFLAG,AL
474: ;
475: MOV YL,0
476: MODE PRINT11,5
477: COPYM,2
478: LOOP_YLT:
479: CMP BYTE PTR CFLAG,0
480: JE C_SKIP
481: CALL TCOPY_R
482: C_SKIP:
483: MOV XL,0
484: MODE BIT8,6
485: LOOP_XL:
486: CALL GC
487: CALL LP88
488: INC XL
489: CMP XL,80
490: JB LOOP_XL
491: ;
492: MOV CL,CR
493: CALL LPRINT
494: MOV CL,LF
495: CALL LPRINT
496: ;
497: INC YL
498: CMP YL,50
499: JB LOOP_YLT
500: ;

```



グラフィックのシステム・コール(ハード・コピ)画面のデータをプリンタ用のデータに変換する

601:	MOV	AX,DS	
602:	MOV	ES,AX	
603:	;		
604:	POP	BX	
605:	POP	AX	
606:	;		
607:	MOV	CL,8	
608:	MOV	CH,8	
609:	MOV	DI,OFFSETDS:BITD8	
610:	;		
611:	PUSH	DS	
612:	MOV	DX,DS.L10	
613:	MOV	DS,DX	
614:	INT	OCEH	
615:	POP	DS	
616:	;		
617:	RET		
618:	GC	ENDP	
619:	;		
620:	PROC	NEAR	
621:	MOV	BX,OFFSETDS:BITD8	
622:	MOV	CX,8	
623:	LOOP		
624:	PUSH	CX	
625:	MOV	CL,[BX]	
626:	CALL	LPRINT	
627:	INC	BX	
628:	POP	CX	
629:	LOOP	LOOP	
630:	RET		
631:	LPB8	ENDP	
632:	;		
633:	PROC	NEAR	
634:	MOV	BX,OFFSETDS:BITD16	
635:	MOV	CX,8	
636:	LOOP		
637:	PUSH	CX	
638:	MOV	CL,[BX]	
639:	MOV	CH,CL	
640:	CALL	LPRINT	
641:	INC	BX	
642:	MOV	CL,[BX]	
643:	MOV	DH,CL	
644:	CALL	LPRINT	
645:	MOV	CL,CH	
646:	CALL	LPRINT	
647:	MOV	CL,DH	
648:	CALL	LPRINT	
649:	INC	BX	
650:	POP	CX	
651:	LOOP	LOOP	
652:	RET		
653:	LPB16	ENDP	
654:	;		
655:	PUBLIC	SCOPLY	
656:	SCOPLY	PROC	
657:	PUSH	BP	
658:	;		
659:	MOV	YL,0	
660:	MODE	PRINT12,5	
661:	LOOP_YL1:		
662:	MOV	XL,0	
663:	MODE	BIT16,6	
664:	LOOP_XL1:		
665:	CALL	GC_1	
666:	CALL	LPB8_1	
667:	INC	XL	
668:	CMP	XL,80	
669:	JB	LOOP_XL1	
670:	;		
671:	MOV	CL,CR	
672:	CALL	LPRINT	
673:	MOV	CL,LF	
674:	CALL	LPRINT	
675:	;		
676:	INC	YL	
677:	CMP	YL,25	
678:	JB	LOOP_YL1	
679:	;		
680:	MODE	PREND,5	
681:	;		
682:	POP	BP	
683:	RET		
684:	SCOPLY	ENDP	
685:	;		
686:	GC_1	PROC	
687:	MOV	AX,XL	
688:	MOV	BX,8	
689:	MUL	BX	
690:	PUSH	AX	
691:	;		
692:	MOV	AX,YL	
693:	MOV	BX,16	
694:	MUL	BX	
695:	PUSH	AX	
696:	;		
697:	MOV	AX,DS	
698:	MOV	ES,AX	
699:	;		
700:	POP	BX	

\* 画面のデータをプリンタ用のデータに変換する

# SCOPLY

グラフィック画面のハード・コピー(1/2  
サイズ)に、グラフィック画面の1ドットをプリンタの  
1ピクセルとする。縦横ともに1/2の大きさ  
になる。



## リスト5-1) グラフィック・サブルーチン(つづき)

701:	POP	AX		751:	CALL	LPRINT	
702:	;			752:	INC	BX	
703:	MOV	CL,8		753:	INC	DI	
704:	MOV	CH,8		754:	POP	CX	
705:	MOV	DI,OFFSETS:BITD8		755:	LOOP	LOOP1	
706:	;			756:	RET		
707:	PUSH	DS		757:	LPB8_1	ENDP	
708:	MOV	DX,DS_L10		758:	;		
709:	MOV	DS,DX		759:	TCOPY_R	PROC	NEAR
710:	INT	OCEH		760:	MOV	DX,0	
711:	POP	DS		761:	MOV	AX,YL	
712:	;			762:	MOV	BX,2	
713:	MOV	AX,XL		763:	DIV	BX	
714:	MOV	BX,8		764:	CMP	DX,0	
715:	MUL	BX		765:	JE	NE_T	
716:	PUSH	AX		766:	RET		
717:	;			767:	NE_T:		
718:	MOV	AX,YL		768:	MOV	CX,0A000H	
719:	MOV	BX,16		769:	MOV	ES,CX	
720:	MUL	BX		770:	MOV	BX,160	
721:	ADD	AX,8		771:	MUL	BX	
722:	PUSH	AX		772:	MOV	SI,AX	
723:	;			773:	MOV	CX,80	
724:	MOV	AX,DS		774:	TC_LOOP:		
725:	MOV	ES,AX		775:	PUSH	CX	
726:	;			776:	MOV	CL,ES:[SI]	
727:	POP	BX		777:	CALL	LPRINT	
728:	POP	AX		778:	INC	SI	
729:	;			779:	INC	SI	
730:	MOV	CL,8		780:	POP	CX	
731:	MOV	CH,8		781:	LOOP	TC_LOOP	
732:	MOV	DI,OFFSETS:BITD8_2		782:	;		
733:	;			783:	MOV	CL,CR	
734:	PUSH	DS		784:	CALL	LPRINT	
735:	MOV	DX,DS_L10		785:	RET		
736:	MOV	DS,DX		786:	TCOPY_R	ENDP	
737:	INT	OCEH		787:	;		
738:	POP	DS		788:	LPRINT	PROC	NEAR
739:	RET			789:	PUSH	AX	
740:	CC_1	ENDP		790:	LPP:	IN	AL,42H
741:	;			791:	AND	AL,04H	
742:	LPB8_1	PROC	NEAR	792:	JE	LPP	
743:	MOV	BX,OFFSETS:BITD8		793:	MOV	AL,CL	
744:	MOV	DI,OFFSETS:BITD8_2		794:	OUT	40H,AL	
745:	MOV	CX,8		795:	MOV	AL,OEH	
746:	LOOP1:			796:	OUT	46H,AL	
747:	PUSH	CX		797:	MOV	AL,OFH	
748:	MOV	CL,[BX]		798:	OUT	46H,AL	
749:	CALL	LPRINT		799:	POP	AX	
750:	MOV	CL,[D1]		800:	RET		

← テキスト1行コピー

← プリンタにデータを1バイト転送する

← プリンタにデータを転送する

801: LPRINT ENDP

```
802: ;
803: ;
804: ;
805: ;
806: CODE ENDS
807: ;
808: DATA SEGMENT PARA 'DATA'
809: LIO_WORK DB 0
810: DS_LIO DW 0
811: WE_LIO DW 0
812: WS_LIO DW 0
813: PA_LIO DW 0
814: SC_LIO DW 0
815: ;
816: ;PARAMETER DB 3,0,0,1
817: SCREEN_D DW 0,0,0,0,0
818: PARAMETER DW 0,0,0,0,0
819: ;
820: TILE_D DB 27 DUP(?)
821: ;
822: PRINT11 DB 1BH,'T16',0DH
823: PRINT12 DB 1BH,'T12',0DH
824: PREND DB 1BH,'T20',0DH
825: COPYM DB 1BH,'D'
826: COPYN DB 1BH,'M'
827: BIT8 DB 1BH,'S0640'
828: BITD DB 1BH,'I1280'
829: BIT16 DB 1BH,'I0640'
830: CR DB 0DH
831: LF DB 0AH
832: BITD8 DB 80UP(?)
833: BITD8_2 DB 80UP(?)
834: BITD16 DB 16 DUP(?)
835: XL DW 0
836: YL DW 0
837: CFLAG DB 0
838: KPON DB 1BH,'>'
839: KPOFF DB 1BH,'J'
840: KFL DB 0
841: ESC DB 1BH
842: KIN DB 4BH
843: KOUT DB 4BH
844: ;
845: ;
846: ;
847: WORK_START DB 300H DUP(?)
848: ;
849: WORK_END DB ?
850: ;
851: ORG 620H
852: DB 0DE0H DUP(?)
853: DATA ENDS
854: END
```

←グラフィックのワーク・エリア  
(未使用領域に各種のデータ領域をとっている)

←以降グラフィックのワーク・エリア

1: COMMENT \*

```
2: ;
3: グラフィック カラー コピープログラム
4: (倍密度)
5: VERSION 1.0
6: FOR PC-9801 & PC-PR201CL
7: ;
8: *
9: DGROUP GROUP DATA
10: ;
11: CODE SEGMENT 'CODE'
12: ASSUME CS:CODE, DS:DGROUP, SS:DGROUP
13: ;
14: VSEG1 EQU 0A800H
15: VSEG2 EQU 0B000H
16: VSEG3 EQU 0B800H
17: ;
18: RCBT MACRO VS, PL
19: MOV AX, VS
20: MOV ES, AX
21: MOV AL, ES:[SI+BX]
22: NOT AL
23: MOV PL, AL
24: ENDM
25: ;
26: MODE MACRO NA, RS
27: LOCAL LOP
28: PUSH CX
29: PUSH BX
30: MOV BX, OFFSET DS:NA
31: MOV CX, RS
32: LOP:
33: PUSH CX
34: MOV CL, [BX]
35: CALL LPRINT
36: INC BX
37: POP CX
38: LOOP LOP
39: POP BX
40: POP CX
41: ENDM
42: ;
43: PUBLIC CLCOPY
44: CLCOPY PROC FAR
45: MOV YL, 0
46: MODE PRINT1, 5
47: LOOP_Y:
48: MOV AX, YL
49: MOV BX, 640
50: MUL BX
```

\* 画面から1バイト・データを取る

←プリンタのモード設定

CLCOPY

グラフィック画面のカラー・ハード・コピーをとる。1ドットをプリンタの4ビット(2x2)としてコピーする。



51: ;	MOV	OFA, AX	101: ;	MODE	YELLOW, 3	
52: ;	WBIM:		102: ;	MODE	BIT, 6	
53: ;	CLD		103: ;	MOV	VSM, VSEG1	
54: ;	MOV	AX, DS	104: ;	CALL	YMC	
55: ;	MOV	ES, AX	105: ;			
56: ;	MOV	DI, OFFSETDS:WBD	106: ;	MODE	MAGENTA, 3	
57: ;	MOV	AX, AX	107: ;	MODE	BIT, 6	
58: ;	XOR	AX, AX	108: ;	MOV	VSM, VSEG3	
59: ;	MOV	CX, 320	109: ;	CALL	YMC	
60: ;	REP	STOSW	110: ;			
61: ;			111: ;	MODE	CYAN, 3	
62: ;	MOV	XL, 0	112: ;	MODE	BIT, 6	
63: ;	MOV	DI, OFFSETDS:WBD	113: ;	MOV	VSM, VSEG2	
64: ;	MOV	WBP, DI	114: ;	CALL	YMC	
65: ;	MOV	SI, OFA	115: ;			
66: ;	LOOP_W:		116: ;	MODE	BLACK, 3	
67: ;	XOR	BX, BX	117: ;	MODE	BIT, 6	
68: ;	MOV	DX, 1	118: ;	MOV	CX, 640	
69: ;	MOV	DI, WBP	119: ;	MOV	DI, OFFSETDS:WBD	
70: ;	MOV	CX, 8	120: ;	LOOP_B:		
71: ;	LOOP_W2:		121: ;	PUSH	CX	
72: ;	RGBT	VSEG1, B	122: ;	MOV	AL, [DI]	
73: ;	RGBT	VSEG2, R	123: ;	CALL	LPL	
74: ;	RGBT	VSEG3, G	124: ;	INC	DI	
75: ;			125: ;	POP	CX	
76: ;	MOV	AL, B	126: ;	LOOP	LOOP_B	
77: ;	AND	AL, R	127: ;			
78: ;	AND	AL, G	128: ;	MOV	BX, OFFSETDS:CR	
79: ;			129: ;	MOV	CL, [BX]	
80: ;	PUSH	CX	130: ;	CALL	LPRINT	
81: ;	MOV	CX, 8	131: ;	MOV	BX, OFFSETDS:LF	
82: ;	LOOP_W1:		132: ;	MOV	CL, [BX]	
83: ;	TEST	AL, 80H	133: ;	CALL	LPRINT	
84: ;	JE	L1	134: ;			
85: ;	ADD	[DI], DX	135: ;	INC	YL	
86: ;	L1:		136: ;	CMP	YL, 50	
87: ;	SHL	AL, 1	137: ;	JB	LOOP_MM	
88: ;	INC	DI	138: ;	MODE	PRINTL, 5	
89: ;	LOOP	LOOP_W1	139: ;	RET		
90: ;	POP	CX	140: ;	LOOP_MM:		
91: ;	MOV	DI, WBP	141: ;	JMP	LOOP_Y	
92: ;	ADD	BX, 50H	142: ;	CLCOPY	ENDP	
93: ;	SHL	DX, 1	143: ;			
94: ;	LOOP	LOOP_W2	144: ;	YMC	PROC	
95: ;	INC	XL	145: ;	MOV	XL, 0	
96: ;	INC	SI	146: ;	MOV	DI, OFFSETDS:WBD	
97: ;	ADD	WBP, 8	147: ;	MOV	WBP, DI	
98: ;	CMP	XL, 80	148: ;	MOV	SI, OFA	
99: ;	JB	LOOP_W	149: ;	MOV	AX, VSM	
100: ;			150: ;	MOV	ES, AX	

```

151: ;
152: LOOP_M:
153: XOR BX,BX
154: MOV DX,1
155: PUSH ES
156: CLD
157: MOV AX,DS
158: MOV ES,AX
159: MOV DI,OFFSETD:BITDATA
160: XOR AX,AX
161: MOV CX,4
162: REP STOSW
163: POP ES
164: MOV CX,8
165: LOOP_2:
166: MOV DI,OFFSETD:BITDATA
167: MOV AL,ES:[SI+BX]
168: PUSH CX
169: MOV CX,8
170: LOOP_1:
171: TEST AL,80H
172: JE ML1
173: ADD [DI],DX
174: ML1:
175: SHL AL,1
176: INC DI
177: LOOP LOOP_1
178: POP CX
179: ADD BX,50H
180: SHL DX,1
181: LOOP LOOP_2
182: ;
183: MOV DI,OFFSETD:BITDATA
184: MOV BX,WBP
185: MOV CX,8
186: LOOP_3:
187: PUSH CX
188: MOV AL,[DI]
189: NOT AL
190: MOV CL,[BX]
191: NOT CL
192: AND AL,CL
193: CALL LPL
194: INC DI
195: INC BX
196: POP CX
197: LOOP LOOP_3
198: ADD WBP,8
199: ;
200: INC XL

```

```

201: INC SI
202: CMP XL,80
203: JB LOOP_M
204: ;
205: MOV BX,OFFSETD:CR
206: MOV CL,[BX]
207: CALL LPRINT
208: RET
209: YMC
210: ;
211: LPL PROC NEAR
212: PUSH BX
213: PUSH AX
214: XOR BX,BX
215: MOV DL,1
216: MOV CX,4
217: LOOP0:
218: TEST AL,01H
219: JE SK
220: ADD BL,DL
221: SHL DL,1
222: ADD BL,DL
223: SHL DL,1
224: JMP SK2
225: SK:
226: SHL DL,1
227: SHL DL,1
228: SK2:
229: SHR AL,1
230: LOOP LOOP0
231: ;
232: MOV DL,1
233: MOV CX,4
234: LOOP1:
235: TEST AL,01H
236: JE SKK
237: ADD BH,DL
238: SHL DL,1
239: ADD BH,DL
240: SHL DL,1
241: JMP SKK2
242: SKK:
243: SHL DL,1
244: SHL DL,1
245: SKK2:
246: SHR AL,1
247: LOOP LOOP1
248: ;
249: MOV CL,BL
250: CALL LPRINT

```

-VRAMのデータをビット・イメージに直してプリンタに送る



```
251: MOV CL,BH
252: CALL LPRINT
253: MOV CL,BL
254: CALL LPRINT
255: MOV CL,BH
256: CALL LPRINT
257: ;
258: POP AX
259: POP BX
260: RET
261: LPL ENDP
262: ;
263: ;
264: LPRINT PROC NEAR
265: PUSH AX
266: LPP: IN AL,42H
267: AND AL,04H
268: JE LPP
269: MOV AL,CL
270: OUT 40H,AL
271: MOV AL,0EH
272: OUT 46H,AL
273: MOV AL,0FH
274: OUT 46H,AL
275: POP AX
276: RET
277: LPRINT ENDP
278: ;
279: CODE ENDS
280: ;
281: DATA SEGMENT 'DATA'
282: WBP DW 0
283: WBD DB 640 DUP(?)
284: BITDATA DB 8DUP(?)
285: VSM DW 0
286: R DB 0
287: G DB 0
288: B DB 0
289: XL DW 0
290: YL DW 0
291: OFA DW 0
292: ;
293: PRINTI DB 1BH,'T12',0DH
294: PRINTE DB 1BH,'T20',0DH
295: MODEC DB 1BH,'D'
296: BIT DB 1BH,'11280'
297: CR DB 0DH
298: LF DB 0AH
299: BLACK DB 1BH,'C0'
300: YELLOW DB 1BH,'C6'
301: MAGENTA DB 1BH,'C3'
302: CYAN DB 1BH,'C5'
303: DATA ENDS
304: ;
305: END
```

←プリンタにデータを1バイト送る

←黒面素用のデータ領域

```
1: COMMENT *
2: コピー サブルーチン（5階調）
3:
4: FOR PC-9801
5:
6: *
7: DGROUP GROUP DATA
8: ;
9: CODE SEGMENT 'CODE'
10: ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
11: ;
12: VSEG1 EQU 0A800H
13: VSEG2 EQU 0B000H
14: VSEG3 EQU 0B800H
15: ;
16: MODE MACRO NA,RS
17: LOCAL LOP
18: PUSH CX
19: PUSH BX
20: MOV BX,OFFSETDS:NA
21: MOV CX,RS
22: LOP:
23: PUSH CX
24: MOV CL,[BX]
25: CALL LPRINT
26: INC BX
27: POP CX
28: LOOP LOP
29: POP BX
30: POP CX
31: ENDM
32: ;
33: BRGC MACRO VSEG
34: MOV AX,VSEG
35: MOV ES,AX
36: MOV AL,ES:[SI]
37: SHL AL,CL
38: ENDM
39: ;
40: PUBLIC COPY5
41: COPY5 PROC FAR
42: PUSH BP
43: MOV BP,SP
44: ;
45: LES SI,6[BP]
46: MOV AX,ES:[SI]
47: MOV CFLAG,AL
48: ;
49: CALL DCOPY_4
50: BP
```

←モードの設定

COPY 5

CALL COPY5(1)

グラフィック画面の色を5階調の濃淡でハードコピーする。グラフィック画面のみ、1のときテキスト画面と合成する。

色と濃淡は次のように対応する。

色 濃淡レベル

0	.....4
1	.....3
2	.....2
3	.....1
4	.....0
5	.....1
6	.....2
7	.....3

グラフィック画面の1ドットをプリンタの4ビット（2×2）とする。

51: RET	4			101: CALL	LPRINT
52: COPY5	ENDP			102: MOV	CL,LF
53: ;				103: CALL	LPRINT
54: DCOPY_4	PROC	NEAR		104: ;	
55: ;				105: INC	YL
56: MOV	YL,0			106: CMP	YL,50
57: MODE	PRINT,5			107: JB	LOOP_YL2
58: LOOP_YL:				108: ;	
59: CMP	BYTE PTR CFLAG,0			109: MODE	PEND,5
60: JE	SK_D4			110: RET	
61: ;	MODE	KPON,2		111: LOOP_YL2:	
62: CALL	TCOPY_R			112: JMP	LOOP_YL
63: SK_D4:				113: DCOPY_4	ENDP
64: MOV	AX,YL			114: ;	
65: MOV	BX,8			115: COL	PROC
66: MUL	BX			116: PUSH	BX
67: MOV	Y,AX			117: PUSH	CX
68: MOV	X,0			118: PUSH	DX
69: MODE	BIT,6			119: ;	
70: ;				120: MOV	AX,Y
71: LOOP_X:				121: MOV	BX,80
72: PUSH	Y			122: MUL	BX
73: MOV	CX,8			123: MOV	SI,AX
74: MOV	BX,OFFSETDS:CLDATA			124: ;	
75: LOOP_Y:				125: MOV	DX,0
76: CALL	COL			126: MOV	AX,X
77: MOV	[BX],AL			127: MOV	BX,8
78: INC	BX			128: DIV	BX
79: INC	Y			129: MOV	CX,DX
80: LOOP	LOOP_Y			130: ADD	SI,AX
81: ;				131: ;	
82: CALL	MDIZA4			132: XOR	BL,BL
83: ;				133: MOV	DL,1
84: MOV	BX,OFFSETDS:BITDATA			134: BRGC	VSEC1
85: MOV	CX,4			135: TEST	AL,80H
86: ;				136: JE	SKB
87: LOOP:				137: ADD	BL,DL
88: CX				138: SKB:	
89: MOV	CL,[BX]			139: BRGC	VSEC2
90: CALL	LPRINT			140: SHL	DL,1
91: INC	BX			141: TEST	AL,80H
92: POP	CX			142: JE	SKR
93: LOOP	LOOPL			143: ADD	BL,DL
94: ;				144: SKR:	
95: POP	Y			145: BRGC	VSEC3
96: INC	X			146: SHL	DL,1
97: CMP	X,640			147: TEST	AL,80H
98: JB	LOOP_X			148: JE	SKG
99: ;				149: ADD	BL,DL
100: MOV	CL,CR			150: SKG:	

←座標 (X, Y) の色を求める。  
AL←



151:	MOV	AL, BL	201:	MOV	AL, [BX]
152:	POP	DX	202:	PUSH	BX
153:	POP	CX	203:	XOR	BX, BX
154:	POP	BX	204:	MOV	BL, AL
155:	RET		205:	MOV	CL, DL
156:	COL	ENDP	206:	SHL	BL, 1
157:	;		207:	;	
158:	;		208:	MOV	AL, [BX+DI]
159:	;		209:	SHL	AL, CL
160:	MDIZA4 PROC	NEAR	210:	ADD	[SI], AL
161:	MOV	AX, DS	211:	MOV	AL, [BX+DI+1]
162:	MOV	ES, AX	212:	SHL	AL, CL
163:	MOV	DI, OFFSETS:BITDATA	213:	ADD	3[SI], AL
164:	XOR	AX, AX	214:	POP	BX
165:	MOV	CX, 4	215:	INC	BX
166:	CLD		216:	INC	DL
167:	REP	STOSB	217:	INC	DL
168:	;		218:	POP	CX
169:	MOV	BX, OFFSETS:CLDATA	219:	LOOP	LOOP_D2
170:	MOV	SI, OFFSETS:BITDATA	220:	RET	
171:	MOV	DI, OFFSETS:DIZAD	221:	MDIZA4 ENDP	
172:	;		222:	;	
173:	XOR	DL, DL	223:	;	
174:	MOV	CX, 4	224:	TCOPY_R PROC	NEAR
175:	LOOP_D1:		225:	MOV	DX, 0
176:	PUSH	CX	226:	MOV	AX, YL
177:	MOV	AL, [BX]	227:	MOV	BX, 2
178:	PUSH	BX	228:	DIV	BX
179:	XOR	BX, BX	229:	CMP	DX, 0
180:	MOV	BL, AL	230:	JE	NE_T
181:	MOV	CL, DL	231:	RET	
182:	SHL	BL, 1	232:	NE_T:	
183:	;		233:	MOV	CX, 0A000H
184:	MOV	AL, [BX+DI]	234:	MOV	ES, CX
185:	SHL	AL, CL	235:	MOV	BX, 160
186:	ADD	[SI], AL	236:	MUL	BX
187:	MOV	AL, [BX+DI+1]	237:	MOV	SI, AX
188:	SHL	AL, CL	238:	MOV	CX, 80
189:	ADD	2[SI], AL	239:	TC_LOOP:	
190:	POP	BX	240:	PUSH	CX
191:	INC	BX	241:	MOV	CL, ES:[SI]
192:	INC	DL	242:	CALL	LPRINT
193:	INC	DL	243:	INC	SI
194:	POP	CX	244:	INC	SI
195:	LOOP	LOOP_D1	245:	POP	CX
196:	;		246:	LOOP	TC_LOOP
197:	XOR	DL, DL	247:	;	
198:	MOV	CX, 4	248:	MOV	CL, CR
199:	LOOP_D2:		249:	CALL	LPRINT
200:	PUSH	CX	250:	RET	

← テキスト画面のコピー

← 縦8ドット分のドットの色をもとにディザ・パターンを作成する。

← 上4ドット分のディザ・パターンを作成する。

← 下4ドット分のディザ・パターンを作成する。

```

251: TCOPY_R ENDP
252: ;
253: ;
254: LPRINT PROC NEAR
255:   PUSH AX
256:   LPP: IN AL,42H
257:   AND AL,04H
258:   JE LPP
259:   MOV AL,CL
260:   OUT 40H,AL
261:   MOV AL,0EH
262:   OUT 46H,AL
263:   MOV AL,OFH
264:   OUT 46H,AL
265:   POP AX
266:   RET
267: LPRINT ENDP
268: ;
269: CODE ENDS
270: ;
271: DATA SEGMENT 'DATA'
272: BITDATA DB 50UP(?)
273: X DW 0
274: Y DW 0
275: YL DW 0
276: CLDATA DB 80UP(?)
277: ;
278: PRINTI DB 1BH,'T12',0DH
279: PREND DB 1BH,'T20',0DH
280: BIT DB 1BH,'I1280'
281: CR DB 0DH
282: LF DB 0AH
283: KPN DB 1BH,'>'
284: CFLAG DB 0
285: ;
286: DIZAD DB 3,3
287: DB 3,1
288: DB 1,3
289: DB 1,2
290: DB 2,1
291: DB 1,0
292: DB 0,2
293: DB 0,0
294: DATA ENDS
295:

```

```

1: COMMENT *
2:   コピー サブルーチン (8階調)
3:
4:   FOR PC-8801
5:
6:   *
7:   DGROUP GROUP DATA
8:   ;
9:   CODE SEGMENT 'CODE'
10:  ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
11:  ;
12:  VSEG1 EQU 0A800H
13:  VSEG2 EQU 0B000H
14:  VSEG3 EQU 0B800H
15:  ;
16:  MODE MACRO NA,RS
17:  LOCAL LOP
18:  PUSH CX
19:  PUSH BX
20:  MOV BX,OFFSETDS:NA
21:  MOV CX,RS
22:  LOP:
23:  PUSH CX
24:  MOV CL,[BX]
25:  CALL LPRINT
26:  INC BX
27:  POP CX
28:  LOOP LOP
29:  POP BX
30:  POP CX
31:  ENDM
32: ;
33: BRGC MACRO VSEG
34:   MOV AX,VSEG
35:   MOV ES,AX
36:   MOV AL,ES:[SI]
37:   SHL AL,CL
38:   ENDM
39: ;
40: PUBLIC COPY8
41: COPY8 PROC FAR
42:   CALL DCOPY_8
43:   RET
44: COPY8 ENDP
45: ;
46: ;
47: DCOPY_8 PROC NEAR
48:   MOV YL,0
49:   MODE PRINTI,5
50: LOOP_YL:

```

# COPY 8

グラフィック画面の色を8階調の濃淡でハード・コピーをとる。

色と濃淡は次のように対応する。

色	濃淡レベル
0	濃
1	濃
2	濃
3	濃
4	濃
5	濃
6	濃
7	濃

グラフィック画面の1ドットをプリンタの9ビット(3×3)とする。



```

51: MOV AX,YL
52: MOV BX,8
53: MUL BX
54: MOV Y,AX
55: MOV X,0
56: MODE BIT,6
57: ;
58: LOOP_X:
59:   Y
60:   MOV CX,8
61:   MOV BX,OFFSETDS:CLDATA
62:   LOOP_Y:
63:     CALL COL
64:     MOV [BX],AL
65:     INC BX
66:     INC Y
67:     LOOP LOOP_Y
68:   ;
69:   CALL MDIZA
70:   CALL LBIT
71:   ;
72:   POP Y
73:   INC X
74:   CMP X,640
75:   JB LOOP_X
76:   ;
77:   MOV CL,CR
78:   CALL LPRINT
79:   MOV CL,LF
80:   CALL LPRINT
81:   ;
82:   INC YL
83:   CMP YL,50
84:   JB LOOP_YL
85:   MODE PREND,5
86:   RET
87: DCOPY_8 ENDP
88: ;
89:   PROC NEAR
90:   PUSH BX
91:   PUSH CX
92:   PUSH DX
93:   ;
94:   MOV AX,Y
95:   MOV BX,80
96:   MUL BX
97:   MOV SI,AX
98:   ;
99:   MOV DX,0
100:  MOV AX,X

```

```

101: MOV BX,8
102: DIV BX
103: MOV CX,DX
104: ADD SI,AX
105: ;
106:   XOR BL,BL
107:   MOV DL,1
108:   BRGC VSEG1
109:   TEST AL,80H
110:   JE SKB
111:   ADD BL,DL
112:   SKB:
113:   BRGC VSEG2
114:   SHL DL,1
115:   TEST AL,80H
116:   JE SKR
117:   ADD BL,DL
118:   SKR:
119:   BRGC VSEG3
120:   SHL DL,1
121:   TEST AL,80H
122:   JE SKG
123:   ADD BL,DL
124:   SKG:
125:   MOV AL,BL
126:   POP DX
127:   POP CX
128:   POP BX
129:   RET
130: COL ENDP
131: ;
132: ;
133: DC
134:   MOV AL,[BX]
135:   PUSH BX
136:   XOR BX,BX
137:   MOV BL,AL
138:   SHL AL,1
139:   ADD BL,AL
140:   ENDM
141: ;
142: DTR
143:   POP BX
144:   INC BX
145:   DC
146:   ENDM
147: ;
148: DSQ
149:   MOV AL,[BX+DI]
150:   MOV CL,SN

```

151:	SHL	AL, CL	DSQ	6, 0, 3, 6	252:	X	DW	0
152:	ADD	[SI+SI], AL	DCC	2, 1, 4, 7	253:	Y	DW	0
153:	MOV	AL, [BX+DI+1]	DTR		254:	YL	DW	0
154:	MOV	CL, SN	203:	DSQ	255:	CLDATA	DB	8DUP(?)
155:	SHL	AL, CL	204:	DSQ	256:	;		
156:	ADD	[SI+SI], AL	205:	DTR	257:	PRINTI	DB	1BH, 'T18', 0DH
157:	MOV	AL, [BX+DI+2]	206:	DSQ	258:	PREND	DB	1BH, 'T20', 0DH
158:	MOV	CL, SN	207:	DTR	259:	BIT	DB	1BH, 'J1920'
159:	SHL	AL, CL	208:	DSQ	260:	CR	DB	0DH
160:	ADD	[SI+SI], AL	209:	DCC	261:	LF	DB	OAH
161:	ENDM		210:	DTR	262:	;		
162:	;		211:	DSQ	263:	DIZAD	DB	7, 7, 7
163:	DCC	MACRO SSN, SS1, SS2, SS3	212:	DTR	264:			6, 5, 3
164:	MOV	AL, [BX+DI]	213:	DSQ	265:	DB		5, 2, 5
165:	MOV	CL, SSN	214:	POP	266:	DB		2, 5, 2
166:	SHR	AL, CL	215:	RET	267:	DB		1, 2, 4
167:	ADD	[SI+SI], AL	216:	MDIZA	268:	DB		1, 2, 0
168:	MOV	AL, [BX+DI+1]	217:	;	269:	DB		0, 2, 0
169:	MOV	CL, SSN	218:	LBIT	270:	DB		0, 0, 0
170:	SHR	AL, CL	219:	MOV	271:	DATA	ENDS	
171:	ADD	[SI+SI], AL	220:	MOV	272:	END		
172:	MOV	CL, SSN	221:	;				
173:	MOV	AL, [BX+DI+2]	222:	LOOPL				
174:	SHR	AL, CL	223:	PUSH				
175:	ADD	[SI+SI], AL	224:	MOV				
176:	ENDM		225:	CALL				
177:	;		226:	INC				
178:	MDIZA	PROC NEAR	227:	POP				
179:	MOV	AX, DS	228:	LOOP				
180:	MOV	ES, AX	229:	RET				
181:	MOV	DI, OFFSETDS:BITDATA	230:	LBIT				
182:	XOR	AX, AX	231:	;				
183:	MOV	CX, 9	232:	;				
184:	CLD		233:	LPRINT				
185:	REP	STOSB	234:	PUSH				
186:	;		235:	LPP				
187:	MOV	BX, OFFSETDS:CLDATA	236:	AND				
188:	MOV	SI, OFFSETDS:BITDATA	237:	JE				
189:	MOV	DI, OFFSETDS:DIZAD	238:	MOV				
190:	;		239:	OUT				
191:	DC		240:	MOV				
192:	MOV	CL, [BX+DI]	241:	OUT				
193:	ADD	[SI], CL	242:	MOV				
194:	MOV	CL, [BX+DI+1]	243:	OUT				
195:	ADD	[SI+3], CL	244:	POP				
196:	MOV	CL, [BX+DI+2]	245:	RET				
197:	ADD	[SI+6], CL	246:	LPRINT				
198:	DTR		247:	;				
199:	DSQ		248:	CODE				
200:			249:	;				
			250:	DATA				
			251:	BITDATA				
				SEGMENT 'DATA'				
				9DUP(?)				



```

1: COMMENT *
2: グラフィック 画面 反転サブルーチン
3:
4:   FOR PC-9801
5: *
6: DGROUP GROUP DATA
7: ;
8: CODE SEGMENT 'CODE'
9: ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
10: ;
11: VSEG1 EQU 0A800H
12: VSEG2 EQU 0B000H
13: VSEG3 EQU 0B800H
14: ;
15: RGBN MACRO VS
16:   MOV AX,VS
17:   MOV ES,AX
18:   MOV AX,ES:[DI]
19:   NOT AX
20:   MOV ES:[DI],AX
21:   ENDM
22: ;
23: ;
24: PUBLIC NEGA
25: NEGA PROC FAR
26:   XOR DI,DI
27: ;
28:   LOOP_MN:
29:     RGBN VSEG1
30:     RGBN VSEG2
31:     RGBN VSEG3
32: ;
33:     INC DI
34:     INC DI
35:     CMP DI,7D00H
36:     JB LOOP_MN
37: ;
38:   RET
39: NEGA ENDP
40: ;
41: ;
42: ;
43:   RGBN MACRO VS2,PL2
44:     MOV CX,VS2
45:     MOV ES,CX
46:     XOR ES:[DI],AX
47:     ENDM
48: ;
49: ;
50:   RGBN MACRO VS,PL

```

NEGA (グラフィック画面の反転)

カラー・モード時では、もとの色とは補色の関係となる。モノクロ・モード時は白黒反転する。アクティブ画面のVRAM全体を対象とする。

```

51:   MOV AX,VS
52:   MOV ES,AX
53:   MOV AX,ES:[DI]
54:   MOV PL,AX
55:   ENDM
56: ;
57: ;
58:   PUBLIC BWNEGA
59: BWNEGA PROC FAR
60:   XOR DI,DI
61: ;
62:   LOOP_MN:
63:     RGBT VSEG1,B
64:     RGBT VSEG2,R
65:     RGBT VSEG3,G
66: ;
67:     MOV AX,B
68:     AND AX,R
69:     AND AX,G
70: ;
71:     MOV BX,B
72:     OR BX,R
73:     OR BX,G
74: ;
75:     NOT BX
76:     OR AX,BX
77: ;
78:     RGBR VSEG1,B
79:     RGBR VSEG2,R
80:     RGBR VSEG3,G
81: ;
82:     INC DI
83:     INC DI
84:     CMP DI,7D00H
85:     JB LOOP_MN
86: ;
87:   RET
88: BWNEGA ENDP
89: CODE ENDS
90: ;
91:   DATA SEGMENT 'DATA'
92:   B DW 0
93:   R DW 0
94:   G DW 0
95:   DATA ENDS
96:   END

```

```

1: COMMENT &
2:
3: BIT search function
4:
5: &
6: DATA SEGMENT 'DATA'
7: DATA ENDS
8:
9: DGROUP GROUP DATA
10: CODE SEGMENT 'CODE'
11: ASSUME CS:CODE,DS:DCGROUP,SS:DCGROUP
12:
13: PUBLIC IBIT
14: IBIT PROC FAR
15: PUSH BP
16: MOV BP,SP
17: LES DI,[BP+10]
18: MOV AX,ES:[DI]
19: LES DI,[BP+6]
20: MOV CX,ES:[DI]
21: AND CL,0FH
22: JZ IB1
23: SHR AX,CL
24: IB1:
25: AND AX,0001H
26: XOR DX,DX
27: POP BP
28: RET 8
29:
30: IBIT ENDP
31:
32: PUBLIC IAND
33: IAND PROC FAR
34: PUSH BP
35: MOV BP,SP
36: LES DI,[BP+10]
37: MOV AX,ES:[DI]
38: LES DI,[BP+6]
39: MOV CX,ES:[DI]
40: AND AX,CX
41: XOR DX,DX
42: POP BP
43: RET 8
44:
45: IAND ENDP
46:
47: PUBLIC IOR
48: IOR PROC FAR
49: PUSH BP
50: MOV BP,SP

```

**IBIT**  
**I = I B I T ( I , J )**  
 2バイト整数のJビット目の値を返す (0 か 1)。  
 2バイト整数に限る。(0 ≤ J ≤ 15)  
 ビットは下位バイトの LSB から、上位バイトの MSB にかけて 0, 1 ..., 15 と数える。

**IAND**  
**I = I A N D ( I , J )**  
 2バイト整数とJの論理積を返す。  
 2バイト整数に限る。

**IOR**  
**I = I O R ( I , J )**  
 2バイト整数とJの論理和を返す。  
 2バイト整数に限る。

**ISHIFT**  
**I = I S H I F T ( I , J )**  
 2バイト整数をJビットシフトした値を返す。  
 2バイト整数に限る。  
 J = 0 : なにもしない。  
 0 < J ≤ 16 : Jビット左にシフト。  
 -16 ≤ J < 0 : Jビット右にシフト。  
 J > 16 : 0を返す。

```

51: LES DI,[BP+10]
52: MOV AX,ES:[DI]
53: LES DI,[BP+6]
54: MOV CX,ES:[DI]
55: OR AX,CX
56: POP BP
57: RET 8
58:
59: IOR ENDP
60:
61: PUBLIC ISHIFT
62: ISHIFT PROC FAR
63: PUSH BP
64: MOV BP,SP
65: LES DI,[BP+10]
66: MOV AX,ES:[DI]
67: LES DI,[BP+6]
68: MOV CX,ES:[DI]
69: CMP CX,0
70: JZ RET_SF
71: JL RSHIFT
72: LSHIFT:
73: CMP CL,16
74: JBE LSH_1
75: MOV CL,16
76: LSH_1:
77: SHL AX,CL
78: JMP SHORT RET_SF
79: RSHIFT:
80: DEC CX
81: NOT CX
82: CMP CL,16
83: JBE RSH_1
84: MOV CL,16
85: RSH_1:
86: SHR AX,CL
87: RET_SF:
88: POP BP
89: RET 8
90: ISHIFT ENDP
91: CODE ENDS
92:

```



```

1: COMMENT *
2: IPEEK & IPOKE
3: IPEEK(SEGMENT, OFFSET)
4: IPOKE(SEGMENT, OFFSET, DATA)
5: ADDRESS SEGMENT : 2バイト整数
6: OFFSET : 2バイト整数
7: DATA : 2バイト整数
8: *
9: DATA SEGMENT 'DATA'
10: DATA ENDS
11: ;
12: DGROUP GROUP DATA
13: ;
14: CODE SEGMENT 'CODE'
15: ASSUME CS:CODE, DS:DGROUP, SS:DGROUP
16: ;
17: PUBLIC IPEEK
18: IPEEK PROC FAR
19: PUSH BP
20: MOV BP, SP
21: ;
22: LES SI, [BP+6]
23: MOV BX, ES:[SI]
24: LES SI, [BP+10]
25: MOV AX, ES:[SI]
26: MOV ES, AX
27: ;
28: MOV AL, ES:[BX]
29: XOR AH, AH
30: XOR DX, DX
31: ;
32: POP BP
33: RET 8
34: IPEEK ENDP
35: ;
36: PUBLIC IPOKE
37: IPOKE PROC FAR
38: PUSH BP
39: MOV BP, SP
40: ;
41: LES SI, [BP+6]
42: MOV CX, ES:[SI]
43: LES SI, [BP+10]
44: MOV BX, ES:[SI]
45: LES SI, [BP+14]
46: MOV AX, ES:[SI]
47: MOV ES, AX
48: MOV ES:[BX], CL
49: ;
50: POP BP
51: RET 12
52: IPOKE ENDP
53: CODE ENDS
54:

```

**IPEEK**  
ID = IPEEK(ISEG, IOFF)  
セグメントとオフセットで指定されたアドレスから1バイトのデータを読み出す。  
引数 { ISEG : セグメント (2バイト整数)  
IOFF : オフセット (2バイト整数) }  
IDの下位にデータが入る、上位は0になる。

**IPOKE**  
CALL IPOKE(ISEG, IOFF, ID)  
セグメントとオフセットで指定されたアドレスに1バイトのデータを書き込む。  
ID : 書き込むデータ  
書き込むデータはIDの下位に入る。

```

1: COMMENT *
2: MS-DOS SYSTEM CALL
3: FOR PC-9801
4: VERSION 1.0
5: *
6: DATA SEGMENT 'DATA'
7: DATA ENDS
8: ;
9: DGROUP GROUP DATA
10: ;
11: CODE SEGMENT 'CODE'
12: ASSUME CS:CODE, DS:DGROUP, SS:DGROUP
13: ;
14: PUBLIC IKEY
15: IKEY PROC FAR
16: MOV AH, 08H
17: INT 21H
18: XOR AH, AH
19: XOR DX, DX
20: RET 4
21: IKEY ENDP
22: ;
23: PUBLIC GTIME
24: GTIME PROC FAR
25: PUSH BP
26: MOV BP, SP
27: ;
28: MOV AH, 2CH
29: INT 21H
30: ;
31: LES SI, 14[BP]
32: MOV ES:[SI], CH
33: MOV BYTE PTR ES:[SI], 0
34: LES SI, 10[BP]
35: MOV ES:[SI], CL
36: MOV BYTE PTR ES:[SI], 0
37: LES SI, 6[BP]
38: MOV ES:[SI], DH
39: MOV BYTE PTR ES:[SI], 0
40: ;
41: POP BP
42: RET 12
43: GTIME ENDP
44: ;
45: PUBLIC GDATE
46: GDATE PROC FAR
47: PUSH BP
48: MOV BP, SP
49: ;
50: MOV AH, 2AH

```

**GTIME (時刻の取り出し)**  
CALL GTIME(IH, IM, IS)  
MS-DOSの管理する時刻を取り出す。  
IH : 時 (0 ~ 23)  
IM : 分 (0 ~ 59)  
IS : 秒 (0 ~ 59)

**IKEY**  
I = IKEY (dummy)  
Iに押されたキーのアスキー・コードが入る。  
dummyはなんでもよい。  
何かキーが押されるまで戻らない。

**GDATE (日付けの取り出し)**  
CALL GDATE(IY, IM, ID, IW)  
MS-DOSの管理する日付けを取り出す。  
IY : 年 (西暦1980 ~ 2079)  
IM : 月 (1 ~ 12)  
ID : 日 (1 ~ 31)  
IW : 曜日 (0 = 日, 1 = 月, ..., 6 = 土)。





## トランジスタ技術 SPECIAL

No. 4 のお知らせ (87年 6 月 25 日発売予定)

### C-MOS標準ロジックIC活用マニュアル

現在、標準ロジックICとしてはTTL74シリーズが有名で、数百品種のICが市販されています。

一般にデジタル回路のハードウェアはこれら標準ロジックICの組み合わせで作られますので、ICの選び方、使い方がシステム設計時には特に重要になってきました。

そこで次号では、TTL74シリーズのC-MOS版74HCシリーズ、標準タイプの4000/4500シリーズなどのC-

MOS標準ロジックICについて詳解します。登場するICは、基本ゲート/ラッチ/フリップフロップ/マルチバイブレータ/カウンタ/デコーダ/エンコーダ/シフトレジスタ/ディスプレイ・ドライバなど、ほとんどすべての機能のICを紹介する予定です。

基本的な機能説明のほか、すぐに活用できる具体的な回路例も豊富です。

ご期待ください。

## 別冊 トランジスタ技術 SPECIAL No. 3

© 1987年 CQ出版社

昭和62年 5 月 1 日発行

定価 1,500円

送料 250円

発行人 飛 坐 博

編集人 蒲 生 良 治

発行所 CQ出版株式会社

〒170 東京都豊島区巣鴨1-14-2

電 話 03(947)6311~6315

振 替 東京 0-10665

印刷・製本 三晃印刷株式会社

# 数値演算プロセッサ

浮動小数点演算入門から高速演算プログラミングまで

インターフェース編集部 編, B5判, 272頁

定価 1,800円, 送料 250円



大型コンピュータで数値計算を行うのはあたりまえだが、マイクロコンピュータで数値計算を行うのは不安だとする人は多い。なぜなら、ソフトあるいはハードの開発者にとっては、マイコンは裸のコンピュータだから、浮動小数点演算などの数学の世界にある程度精通してなくてはならないからだろう。ところがIEEEでは、マイコンが今後、数値計算の世界でも大きなシェアを占めることを予想して、数値計算の専門家などがこの浮動小数点演算を規格化(IEEE-754)した。この規格に準拠していれば、最新の数学的理念に裏付けされた、しかも移植性の高い数値計算の世界が開かれるのである。

本書では、浮動小数点演算の入門から、このIEEE規格、その規格に準拠した各種プロセッサの活用法までを、詳細にわかりやすく解説している。これから、ロボットなどの制御、グラフィックス、いろいろな解析などで、数値計算を行わなくてはならない人にとって必読の書であろう。

## ■本書の内容■

### 第1部 パソコン数値計算入門

- 第1章 浮動小数点演算入門
- 第2章 パソコン数値計算における誤差と対策
- 第3章 パソコン用BASICにおける数学関数の誤差解析
- 第4章 IEEE浮動小数点演算規格の概要と背景

### 第2部 数値演算コプロセッサi8087の研究

- 第5章 8087/287/387活用のための基礎知識
  - 付録 各種コンパイラで8087使用法
- 第6章 アセンブラによる8087の効果的活用法
- 第7章 CとFortranにおける8087の支援と評価
- 第8章 C言語プログラムにおける8087インライン・サポート関数の作り方
- 第9章 高速・高精度行列演算のための8087活用法

- 第10章 8087を用いた高速フーリエ変換プログラム
- 第11章 マクロ・アセンブラを用いた8087数値演算(基本関数)ライブラリ

### 第3部 IEEE準拠演算プロセッサの研究

- 第12章 MC68020用浮動小数点コプロセッサMC68881の概要と活用法
- 第13章 NS32000用浮動小数点スレーブ・プロセッサNS32081の概要と活用法
- 第14章 V20/30/40/50用浮動小数点プロセッサμPD72191の概要
- 第15章 高速64ビット浮動小数点乗算器/ALU ADSP3210/3220の概要と活用法
- 第16章 高速64ビット浮動小数点乗算器/ALU WTL1164/1165の概要と活用法



好評発売中

# 科学計測のための 波形データ処理

— 計測システムにおけるマイコン/パソコン活用技術 —

南 茂 夫 編著

A 5 判, 240 頁, 定価 1,900 円

## ■本書の特徴■

実験室や計測現場では、センサから得られた生の波形データを、雑音除去などの信号処理を施したり、周波数分析など解析したりすることが多くあります。本書では、計測データの波形処理として使われる各手法について、その基礎から実際の応用まで、やさしくかつ具体的に解説しています。マイコン技術者の方だけでなく、物理・化学・生物など実験室の研究者の方にとっても、必携の書といえます。

なお、本書で示したプログラムは、実際に実験室で利用中のもので、そのほとんどがPC-9801用Basicで作成されています。

## ■本書の内容■

- 第1章 科学計測とラボラトリ・オートメーション
- 第2章 波形データとマイコン/パソコン
- 第3章 アナログ入出力とデータ変換
- 第4章 不規則雑音の解析
- 第5章 演算処理による雑音除去法
- 第6章 信号波形の検出と抽出
- 第7章 波形歪の補正法(ディコンボリューション処理)
- 第8章 フーリエ変換
- 第9章 最大エントロピー法
- 第10章 重畳波形の分離と分解
- 第11章 多変量解析手法

## ■本書に掲載したプログラム■

### 〈雑音解析用〉

- 実時間振幅分布測定プログラム(8085用アセンブラ)
- 非実時間振幅分布測定プログラム(Basic)
- 自己相関関数演算プログラム(Basic)

### 〈雑音除去用〉

- 2次・3次多項式適合平滑化プログラム(Basic)
- 適応化平滑化法プログラム(Basic)
- 実時間演算平均処理プログラム(8085用アセンブラ)

### 〈信号波形の検出/抽出用〉

- ピーク検出プログラム(Basic)
- 相互相関関数計算プログラム(Basic)

### 〈信号源波形推定用〉

- ディコンボリューション処理プログラム(Basic)

### 〈高速フーリエ変換用〉

- Sande-Tukey法によるFFT演算プログラム(Basic)
- Sande-Tukey法によるFFT演算プログラム(8086用アセンブラ)
- ビット逆転/sin/cosテーブル作成プログラム(Basic)
- FFT演算プログラム例(Basic)

### 〈最大エントロピー法用〉

- MEMプログラム例(Basic)

### 〈波形分解・分離処理用〉

- 初期パラメータ決定用プログラム(Basic)
- シンプレックス法による波形分離プログラム(Basic)
- DFP法による波形分離プログラム(Basic)
- Gauss-Newton法による波形分離プログラム(Basic)

### 〈重回帰分析法用〉

- 非負拘束つき最小2乗法プログラム(Basic)
- 固有値解析プログラム(Basic)



# モリモリハードディスク

## 自己増殖型ユニット S-DISK



### S-DISK PACK

バック特別定価  
¥62,000

次から次へと便利な物が出来てしまう今日此の頃ではありますが、それにしてもS-DISKにはマイツいてしまいます。いかい便利やねえと思つたところへ、さらに機能を追加する言うのですからたまりません。まず最近流行のキャッシュディスク機能はオートアップデートまでサポートしたスグレモノ。唯でさえ高速アクセスになったところへ専用オプティマイザーまで装備してしまつたもので、すから、これはもうとんでもない速さでハードディスクを使い込めるというわけです。このたび最大の呼び物、仮想ディスク。初登場のこの機能は、ディスクをより効率的に使用することにより収納出来るデータ量のアップを計ろうというもの。簡単に言えば10MBのハードディスクを15MBや30MB、CGデータなんかの場合だと100MB以上の大容量ハードディスクに変身させてしまうというわけなのです。しかもこれだけ機能を追加しておいて価格は据え置きだと言うのですから、なかなか見上げた根性だといふべきでしょう。ハードディスクをモリモリしたいあなたへ、ちょっと素敵なお知らせです。

P. S ハードディスク以外になんとフロッピーディスク、メモリーディスク、更には光ディスクなんかも仮想ディスク機能が發揮されてしまうのです。コワイデスネ。

"グレートベン"さりげなく新登場!

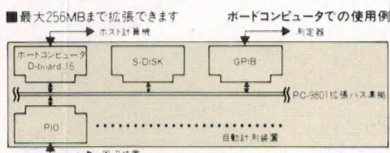
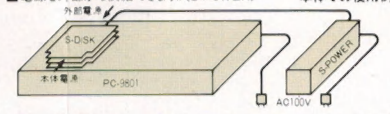
#### S-DISK PACK内容

- S-DISK (1MB) ×1 ■S-POWER ×1
  - 専用ドライバソフト (ベンジャミン BENJAMIN) 通称グレートベン
- バック特別定価 ¥62,000

#### 特徴

1. パソコン本体の電源を切っても、記録データを保存します。電源は専用外部電源S-POWERの接続で、自動的にバックアップされます。
2. 従来のメモリーディスクと違い、完全なI/O装置として稼働するため、ディスク内転送アドレスのハードウェア化による高速アクセスを実現しました。パソコンのメモリー空間はプログラムのために使用してください。
3. S-DISKの高速性能で一般のディスクを使用できるキャッシュ機能。従来のビフォーアード、ライトスルー等の機能に加えオートアップデート機能もサポートされています。
4. 今回最大の追加機能がこの仮想ディスク機能。ディスクの容量を見掛け上増加させるのです。MS-DOSのコマンドで10%~20% (コマンドによっては50%UPも可能)、CGデータ等の場合なら、300%~1,000%以上の効率向上が可能です。
5. 将来を考えて、優れた拡張機能をもたせました。最大256MBまで拡張可能、しかもアクセスタイムは変化しません。もし多数の拡張ボードを装着して電源容量がオーバーしようときも、外部から電源を供給できますので心配はいりません。また、それぞれのS-DISKの状態は発光ダイオードにより一目で確認出来ます。

- PC-9801E/F/M/VM/VF/U/UV/VM21/VX/XL/XA/D-board 16にて使用できます。
- リフレッシュコントロールが不要です。(PC-9801から外して使用出来ます。)
- 電源を外部から供給できます。(S-POWER) 本体での使用例



記憶容量

1MB/512KB 256MBまで拡張可能。

#### 応用例

- ワープロなど、頻繁にディスクを使用する装置での不揮発の高速大容量ディスクとして。
- ディスクを使用するプログラムの高速化。
- メインメモリーが、メモリーディスクとして使用できないシステムの高速ディスクとして。
- 大量の高速ディスクが必要な、画像処理など。
- 自動計測装置のリアルタイムディスク装置として。
- LANなどの高速ディスクサーバーとして。
- 振動など、設置条件により従来の可動式ディスクが使用出来ないシステム。
- D-board16と組合わせた、インテリジェントディスク装置。

#### LINE UP

- S-DISK (1MB) ..... ¥60,000
- S-DISK-512 (512KB) ..... ¥50,000
- S-POWER (専用外部電源) ..... ¥12,000
- BENJAMIN 5インチ2HD MS-DOSバージョン3.1 ..... ¥15,000

コンピュータを通じてあなたの夢をかなえる

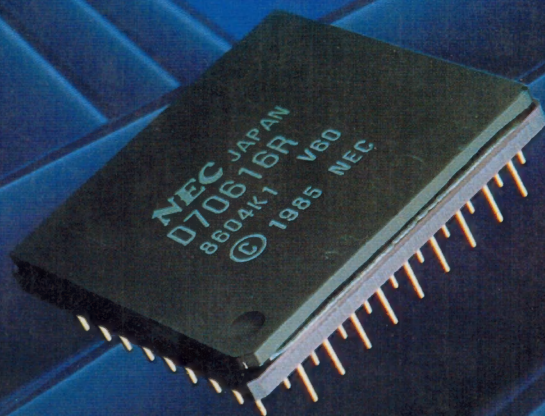


(株)コンピュータドリームデベロップメント

〒559 大阪市住之江区安立3丁目9-13  
TEL. (06) 675-5861 FAX. (06) 675-5862

●只今販売代理店募集中 / 詳細はお電話で。



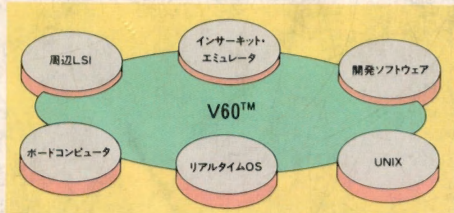


NEC32ビット・オリジナル・マイクロプロセッサμPD70616(V60™)

## V60™のシステム開発を強力にバックアップ。

自主開発による国産初の32ビット・オリジナル・マイクロプロセッサ(μPD70616(V60™))。1チップに375,000素子を集積、最大命令処理速度3.5MIPSとスーパーミニコンに匹敵する高性能を発揮します。まさにマイコンの概念を一新するV60™は、次世代システムの中心デバイスとして注目されています。このV60™による応用システムの開発のために、NECはクロックジェネレータ\*μPD71611C、システムコントローラ\*μPD71613Cなどの周辺LSIやインサートキット・エミュレータ\*IE-70616-A016、それに開発ソフトウェア\*Cコンパイラ・パッケージを用意。さらに順次周辺LSIの拡充を図るとともに、各種システムボードやOSとして"UNIX"や"リアルタイムOS"を提供する予定です。"Vシリーズ™"は、開発サポートについても総力をあげて取り組んでいます。ご注目ください。

### V60™開発環境

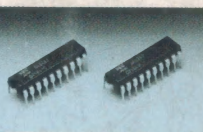
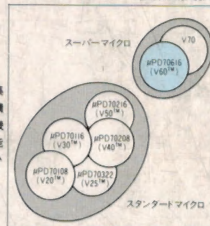


Vシリーズ™、V20™、V30™、V40™、V50™、V25™、V60™は、日本電気株式会社の商標です

### V60™の主な特長

- 仮想記憶管理機能を内蔵
- 浮動小数点演算機能を内蔵
- 高級言語指向の強力な命令セット
- 6段パイプライン方式による命令実行の高速化
- OSサポート機能
- デバッグ・サポート機能
- FRMによる冗長監視系機能
- V20™/V30™エミュレーション機能
- CMOS
- クロック周波数: 16MHz
- 5V単一電源
- 68pin PGAパッケージ

### "Vシリーズ™"製品展開



▲クロックジェネレータ\*μPD71611C  
システムコントローラ\*μPD71613C

▲インサートキットエミュレータ  
\*IE-70616-A016

NECオリジナルマイクロプロセッサ

Vシリーズ™

日本電気株式会社

お問い合わせは: 半導体第一販売事業部、半導体第二販売事業部

半導体市場開発本部第一応用技術部

〒108 東京都港区芝五丁目29-11(日本電気住生ビル) ☎(03)456-6111(大代表)

半導体市場開発本部第二応用技術部

〒530 大阪市北区東島浜一丁目2-6(新大阪ビル) ☎(06)348-1477

半導体応用技術本部マイクロコンピュータメモリ技術部

〒210 川崎市幸区塚越3-484(川崎技術センター) ☎(044)533-1111(大代表)